NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A PROGRAM INTERFACE PROTOTYPE
FOR A MULTIMEDIA DATABASE
INCORPORATING IMAGES

by

Cathy Anne Thomas

December 1988

Thesis Advisor: C. Thomas Wu
Co-Advisor: Klaus Meyer-Wegener

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School — 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A Program Interface Prototype for a Multimedia Database Incorporating Images (Unclassified)

12. PERSONAL AUTHOR(S)
Thomas, Cathy A.

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	December 1988	121

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

17. COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Multimedia Database Management System, Multimedia Database Management System, DBMS, Image Database

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis represents one aspect of an exploration of the integration of unformatted data types, such as image, sound and signal, with more conventional formatted types in a single database. The focus of this thesis is the implementation of a prototype of a database employing a relational model that incorporates both formatted and unformatted data types. Initial research was limited to integration of image data. The prototype provides storage and retrieval capabilities, as well as a modest query-handling capability.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Professor C. Thomas Wu	22b TELEPHONE (Include Area Code) (408)646-3391 — 22c. OFFICE SYMBOL 52Wq

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

☆ U.S. Government Printing Office: 1986—606

A Program Interface Prototype for a Multimedia Database Incorporating Images

by

Cathy Anne Thomas
Lieutenant, United States Navy
B.A., Central Michigan University, 1974
M.S.L., Western Michigan University, 1979

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1988

ABSTRACT

This thesis represents one aspect of an exploration of the integration of unformatted data types, such as image, sound and signal, with more conventional formatted types in a single database. The focus of this thesis is the implementation of a prototype of a database employing a relational model that incorporates both formatted and unformatted data types. Initial research was limited to integration of image data. The prototype provides storage and retrieval capabilities, as well as a modest query-handling capability.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

This thesis represents one portion of the research being conducted on a larger scope, the purpose of which is to explore the integration of unformatted data types, such as image, sound and signal, with more conventional formatted types in a single database. A constraint placed on the project by the sponsor, Naval Ocean Systems Center (NOSC), i.e., to demonstrate within one year a capability to fulfill this purpose sufficient to warrant continued research, affected key implementation decisions and created the need for the work that forms the material for this thesis.

To narrow the scope, initial research was limited to integration of a single unformatted data type, namely image. Examination of the integration of sound data has subsequently been initiated and is the subject of a related thesis by LCDR G. Sawyer (Sawyer, 1988). Also due to the time constraint, it was necessary to utilize existing technology in the implementation effort, another constraint in itself, which dictated many subsequent design decisions, as discussed below.

The focus of this thesis is the implementation of a prototype of a database that incorporates both formatted and unformatted data types. It provides storage and retrieval capabilities, as well as a modest query-handling capability.

B. THE ENVIRONMENT

Selection of the hardware and support software were very pragmatic decisions, driven primarily by on-hand availability. Thus, the system resides on a Sun 3 Workstation, the chosen operating system is Sun OS, similar to Berkeley UNIX 4.3, the programming language of choice is C for interfacing to Relational Technology's INGRES/SQL (version 5.0) database management system, and the Sun Microsystems, Inc. workstation and Pixrect facilities provide the interface for image capture and display. Since each of these tools were already available in-house, this environment offered the additional advantage that many, if not all, of them were already familiar to project participants, further expediting the development effort.

Environment notwithstanding, every effort has been made to keep the prototype implementation independent with an eye to future portability. At present, for instance, the environment described above, though capable of integrating sound data, does not provide support for input and output of sound, so an IBM/MS-DOS alternative is being explored. This environment still utilizes C and INGRES, so software developed for the prototype can be ported directly. It is hoped that the prototype can be kept free of implementation specifics so that it can be adapted to that, or any other selected, environment.

C. SELECTION OF THE RELATIONAL MODEL

The relational model was selected primarily because the database issues have already been dealt with and it is widely understood. This enabled the project participants to concentrate on the multimedia aspects of the problem.

The relational model was not the only viable model to choose from. An important alternative was an object-oriented database management system (DBMS). The hallmarks of the objected-oriented model, namely encapsulation, inheritance and polymorphism, make such a model particularly adaptable to use for multimedia data. These features obviate the necessity for a homogeneous record structure, which is very hard to derive for unformatted data but is required for effective implementation of a model such as the relational one. (Enbody, 1988, pp. 15-19)

Unfortunately, developments in the construction of an object-oriented DBMS are not yet stable, nor is there a commercially available fully functional object-oriented database system. Nonetheless, the object-oriented model seems a desirable one for multimedia applications, and the ability to convert to the object-oriented model in the future was a specific goal of this prototype.

II. MULTIMEDIA DATA IN THE RELATIONAL MODEL

A. METHODS OF REPRESENTATION

Having selected the relational model, there were two primary alternatives to consider for its implementation (Tang, 1980, p. 159).

1. Use of a Special Relation

The first would be to implement a special relation in which each tuple stands for an entity external to the system. This approach offers the advantage that operations can be applied to such tuples that can not be applied to normal tuples, such as "display" in the case of an image. However, it offers some thorny problems which more than offset this benefit, such as that of how to implement a join of tuples where some of them have image attributes and others have sound attributes, or similarly how to implement a projection.

For example, assume an image relation with attributes day and hour. If you were to project on hour, thereby eliminating duplicates and in effect collapsing tuples with the same value into a single tuple, which image does the resulting tuple stand for? All of them? One of them? Any of them? Rather than attach an arbitrary meaning to such a result to resolve such ambiguity, the straight-forward answer would be to disallow projection altogether, but this is obviously not a desirable solution.

2. Use of a Special Attribute

The second alternative, adopted for this implementation, it to introduce a special attribute to represent an external entity such as an image. The projection problems are eliminated. However, a join or a selection based on a comparison of values introduces the question of establishing the properties of equality/inequality of instances of the new type.

B. USE OF A USER-DEFINED ABSTRACT DATA TYPE

The accepted response to this question is to create an abstract data type (ADT) with its own set of operations to access occurrences of the ADT. This implementation occurs at a low level. At higher (user) levels, implementation dependence on the details of how an occurrence of the ADT is stored and maintained is avoided, because such details are handled within the ADT. This solution requires a definition of the functions required to manipulate an occurrence of the ADT. (Ong et al, 1984, p. 2)

Unfortunately, there is no stable, commercially available system that allows for ADT definition. POSTGRES, an INGRES successor, provides the ability to define new ADT's, however, it is not yet on the market. When such a capability is made available, the implementation of the multimedia database using the relational model most likely will be simplified. In the absence of such a capability, the integration of an ADT of type image will be provided by introducing an

additional layer on top of INGRES that accesses both an INGRES DBMS and standard files in order to process queries. See Figure 1. (Meyer-Wegener, 1988, p. 18)

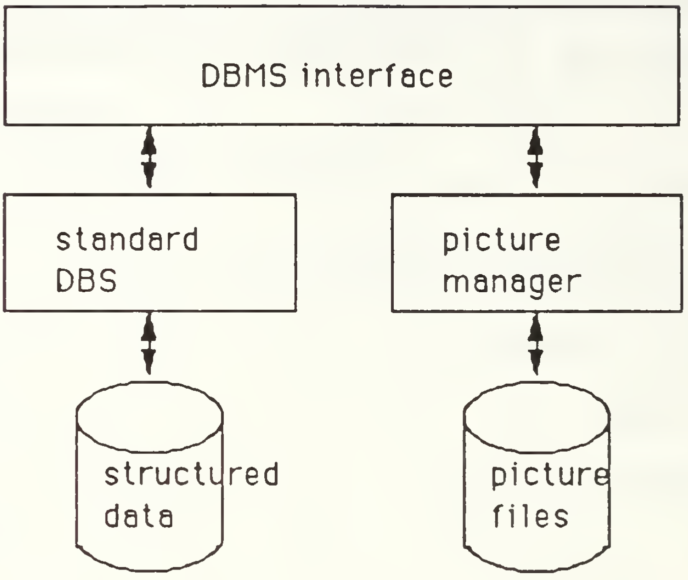

Figure 1. Architecture of the Prototype

III. IMAGE ABSTRACT DATA TYPE

In the context of the above discussion, an IMAGE ADT will be useful for manipulating an image. The operations of the ADT will be used to insert or access IMAGE occurrences in the database.

A. DATA STRUCTURE

In general terms, there are two methods in which a user might want to access a database. Operating in an interactive mode, the user might enter a query at the keyboard and expect an immediate response on the output device, eg., a CRT. Alternatively, by employing a computer program (subsequently referred to as the "program mode"), the program will issue a query to obtain data or to insert data into the database, in which case displaying results is not expected or required.

The inherent difference between these two modes is whether or not a response will have to be displayed. This feature becomes most important in a multimedia application. Consider a request that returns an occurrence of the IMAGE type. In an interactive mode, the user will probably expect to see a visual display of the image itself. However, in the program mode, the technical aspects of image display do not have to be considered; it is sufficient to return the information that is used to construct the image. This difference must be considered in defining the operations of

the ADT, since some operations may be appropriate for one mode but not for the other.

B. OPERATIONS

The "abstract" in "abstract data type" comes from the fact that the user knows what data he can manipulate with the structure and what he has to do to access an occurrence of the structure, but he has no knowledge of how the data is maintained internal to the system. This is one source of hardware independence of a system employing ADT's.

In order to maintain this division between the what and the how, the user is allowed to access occurrences of the ADT only through a set of carefully pre-defined operations. The user knows what data he must provide to the function that implements the operation, and the type of value that the function returns.

C. EXTERNAL FUNCTIONS

In the case of the IMAGE ADT, there are 16 operations available to the user for manipulating an IMAGE occurrence. These are available to the user in the form of what will be termed "external functions" as summarized in Table 1. Note that the external function name is all in upper case.

Note also that there are two functions, IMAGE_FROM_PIXRECT and CONSTRUCT_IMAGE, which may be employed to obtain a new IMAGE value. There is a significant difference between them.

TABLE 1. EXTERNAL FUNCTIONS

Operation	Inputs	Result Type
CONSTRUCT_IMAGE	width, height, depth, encoding, colormap length, entry length, colormap, pixelmatrix	IMAGE
IMAGE_FROM_PIXRECT	pixrect, colormap	IMAGE
PIXRECT	IMAGE	pixrect
HEIGHT	IMAGE	integer
WIDTH	IMAGE	integer
DEPTH	IMAGE	integer
ENCODING	IMAGE	integer
COLORMAP_LENGTH	IMAGE	integer
COLORMAP_ENTRY_SIZE	IMAGE	integer
COLORMAP	IMAGE	colormap
PIXELMATRIX	IMAGE	pixelmatrix
WINDOW	IMAGE, x, y, dx, dy	IMAGE
ADD_DESCRIPTION	IMAGE, description	IMAGE
REPLACE_DESCRIPTION	IMAGE, description	IMAGE
DESCRIPTION_LENGTH	IMAGE	integer
DESCRIPTION	IMAGE	char
SHOWS	IMAGE, pattern	boolean

The first, IMAGE_FROM_PIXRECT, is environment specific in that it expects as input a pixrect a data format known only to the Sun Microsystems Pixrect facilities. In this structure, all of the descriptive information required for the system to construct an image precedes the pixelmatrix that comprises the remainder of the pixrect structure. This is, in effect, the short-cut method of obtaining an IMAGE.

The alternative operation, CONSTRUCT_IMAGE, is intended to be environment independent. Instead of a pixrect, it expects as inputs each of the separate pieces of information that are provided by a pixrect header (height, width, depth, encoding, etc.) and utilizes them, together with the pixelmatrix and colormap provided by the user, to construct an IMAGE. Of course, when employing this operation, the responsibility falls on the user to provide the correct information, as the ADT has no way of verifying correctness of most features of the input.

It will be helpful to examine some sample statements that demonstrate how a user might incorporate the ADT operations into an SQL statement. The first inserts a new image into a relation. Consider a table defined as follows:

```
EXEC SQL CREATE TABLE sample_table (image_id integer,
                                    image_pict IMAGE).
```

To enter an image value into the table, the following statement might be used. (A colon preceding a term signals to INGRES that this is a host variable.)

```
EXEC SQL INSERT INTO sample_table
    (image_id, image_pict)
    VALUES(:curr_id_num,
           IMAGE_FROM_PIXRECT(sample_pixrect,
                              sample_colormap));
```

The next statement selects all images that meet a specified selection criterion. This statement employs the SHOWS external function, which accepts an image attribute and a character string, and returns TRUE or FALSE, depending on whether or not the string is found anywhere in the description value for the associated image.

```
EXEC SQL SELECT image_id,
                PIXRECT(image_pict),
                COLORMAP(image_pict)
    INTO  :sample_ships, :pr, :cm
    FROM  sample_table
    WHERE SHOWS(image_pict,"ship");
```

The last example represents an update to an existing database value.

```
EXEC SQL UPDATE sample_table
    SET image_pict =
        IMAGE_FROM_PIXRECT(new_pixrect,new_colormap)
    WHERE SHOWS(image_pict,"CVN-71");
```

The library of ADT operators can be dynamic, growing or shrinking as a requirement for a new capability is identified or an existing capability is determined to be unnecessary. This characteristic is particularly valuable during the design phase, and can be expected to be utilized most in the prototype, while gaining skill in operation identification and definition.

The reader should be able to see, upon a casual review of the list of operations, that not all of them would be

appropriate in an interactive environment and that operations of particular value in the interactive mode are absent from the list. For instance, an interactive user probably would not be interested in viewing the results of PIXELMATRIX or COLORMAP, for the resulting stream of data would have no meaning when displayed on the screen. On the other hand, it is easy to suppose that an interactive user might desire a DISPLAY operation for an instance of the IMAGE ADT, but such an operation would be useless in the program mode.

For the remainder of this discussion, the focus will be on the program mode only. This further narrows the scope of the project, supporting the desire for rapid development and early demonstration.

D. INTERNAL REPRESENTATION

In application, a user, generally an applications programmer, will manipulate instances of the IMAGE ADT 1) by declaring an attribute to be of type IMAGE, 2) by employing a function call that is expected to return a value of type IMAGE, or 3) by making a function call with a parameter of type IMAGE.

The IMAGE type is undefined to standard INGRES SQL. This is where the additional layer that resides one level above INGRES in the software hierarchy becomes necessary. It is implemented in the form of a preprocessor, translating the external representation of such a structure into a pair of declarations that <u>are</u> recognized by standard INGRES SQL,

namely a filename that represents the actual storage location of the image data and a text description of the contents of the image.

Names in INGRES SQL are sequences of no more than 12 characters. With the exception of table names, all 12 characters must form a unique combination. (Table names require only that the first nine do so.) The following name translation convention is employed. All characters from the user-supplied image name up to a maximum of ten are preserved. To identify the associated filename, an extension of "_f" is attached to this string (or substring), and to identify the description, an extension of "_d" is added. For instance, if the user codes

```
EXEC SQL CREATE TABLE my_table
    (ship_name c20,
     ship_image IMAGE);
```

the preprocessor will translate this to

```
EXEC SQL CREATE TABLE my_table
    (ship_name c20,
     ship_image_f vchar(FILE_NAME_LENGTH),
     ship_image_d vchar(DESCR_LENGTH));
```

where FILE_NAME_LENGTH and DESCR_LENGTH are preprocessor defined constants. It will be the user's responsibility to ensure that the image attribute names contain unique combinations in the first ten characters.

E. INTERNAL FUNCTIONS

Corresponding to each of the external functions is an "internal function" with a similar name. The name is derived

by converting the name of the external function to lower case and prepending the letters "IS." The inputs are the same, unless of type IMAGE, where the image name is translated to a filename/description pair as described above. Unlike the external functions, the internal functions are used more in the sense of a procedure than a function. Thus, the return value becomes an output parameter of the same type, or a pair of parameters in the case of IMAGE type. The return value is reserved for error code passing. See Table 2.

Following through with the same examples utilized for external function usage in Section C above, the statements below show how the same information can be obtained utilizing the internal functions. First is the example of a database insertion.

```
EXEC SQL BEGIN DECLARE SECTION;
    char ISfn1[FILE_NAME_LENGTH + 1];
    char ISdescr1[DESCR_LENGTH + 1];
EXEC SQL END DECLARE SECTION;

ISimage_from_pixrect(sample_pixrect, &sample_colormap,
                     ISfn1, ISdescr1);

EXEC SQL INSERT INTO sample_table
    (image_id, image_pict_f, image_pict_d)
    VALUES (:curr_id_num, :ISfn1, :ISdescr1);
```

Next is an example of a select query.

```
EXEC SQL BEGIN DECLARE SECTION;
    char ISfn2[FILE_NAME_LENGTH + 1];
    char ISdescr2[DESCR_LENGTH + 1];
    int ISvar3;
EXEC SQL END DECLARE SECTION;
```

TABLE 2. INTERNAL FUNCTIONS

Operation	Inputs	Outputs
ISconstruct_image	width, height, depth, encoding, colormap length, entry length, colormap, pixelmatrix	image_f,image_d
ISimage_from_pixrect	pixrect, colormap	image_f,image_d
ISpixrect	image_f,image_d	pixrect
ISheight	image_f,image_d	integer
ISwidth	image_f,image_d	integer
ISdepth	image_f,image_d	integer
ISencoding	image_f,image_d	integer
IScolormap_length	image_f,image_d	integer
IScolormap_entry_size	image_f,image_d	integer
IScolormap	image_f,image_d	colormap
ISpixelmatrix	image_f,image_d,	pixelmatrix
ISwindow	image_f,image_d, x, y, dx, dy	image_f,image_d
ISadd_description	image_f,image_d, description	image_f,image_d
ISreplace_description	image_f,image_d description	image_f,image_d
ISdescription_length	image_f,image_d	integer
ISdescription	image_f,image_d	char
ISshows	image_f,image_d, pattern	boolean

```
EXEC SQL DECLARE CURSOR ISc_one AS
    SELECT image_pict_f, image_pict_d
    FROM sample_table;

EXEC SQL OPEN ISc_one;

EXEC SQL WHENEVER NOT FOUND GOTO IScloseISc_one;

for(;;)
    {
        EXEC SQL FETCH ISc_one
            INTO :ISfn2, :ISdescr2;

        ISshows(ISfn2,ISdescr2,"ship",&ISvar3);

        if (ISvar3)
            {
                ISpixrect(ISfn2,ISdescr2,&pr);
                IScolormap(ISfn2,ISdescr2,&cm);
            }
    }

IScloseISc_one:
    EXEC SQL CLOSE ISc_one;
    EXEC SQL WHENEVER NOT FOUND <old value>;
```

And finally, the example of a database update.

```
EXEC SQL BEGIN DECLARE SECTION;
    char ISfn4[FILE_NAME_LENGTH + 1];
    char ISdescr4[DESCR_LENGTH + 1];
    int ISvar5;
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE CURSOR ISc_one AS
    SELECT image_pict_f, image_pict_d
    FROM sample_table;

EXEC SQL OPEN ISc_one;

EXEC SQL WHENEVER NOT FOUND GOTO IScloseISc_one;

for(;;)

    {
        EXEC SQL FETCH ISc_one
            INTO :ISfn4, :ISdescr4;

        IS shows(ISfn4,ISdescr4,"CVN-71",&ISvar5);

        if (ISvar5)
            {
```

```
            ISimage_from_pixrect
                (new_pixrect,&new_colormap,ISfn4,
                ISdescr4);

        EXEC SQL UPDATE sample_table
            SET image_pict_f = ISfn4,
                image_pict_d = ISdescr4
            WHERE CURRENT OF ISc_one;
             }
    }

ISclosEISc_one:
    EXEC SQL CLOSE ISc_one;
    EXEC SQL WHENEVER NOT FOUND <old value>;
```

IV. THE PREPROCESSOR

A. PURPOSE

The preprocessor implements an SQL-based query language that incorporates the ADT IMAGE. The resulting query language is referred to as IMAGE SQL (ISQL). It provides to the user all of the functionality of a DBMS with integrated image management facilities by establishing a link between a standard relational DBMS, which manages structured data, and a picture manager, which stores images in standard files. These two pieces of software are otherwise unable to communicate or cooperate.

B. INPUTS AND OUTPUTS

The input to the preprocessor is a file with a ".isc" extension containing a C program which may contain statements in either or both INGRES Embedded SQL (ESQL) and ISQL. The preprocessor also requires a file named "function.isql" containing information necessary for the translation of an ISQL statement into its ESQL equivalent. The use of this file is discussed below. The output from the preprocessor is a file of the same name but bearing a ".sc" extension which contains a C program from which all ISQL statements have been removed by substituting appropriate ESQL declarations and statements. This file is in the proper format for submission

to the ESQL preprocessor, preparatory to compilation and linking.

C. FUNCTION TRANSLATION TABLE

The function translation table is built from information contained in the **function.isql** file mentioned above. The correctness of the input is crucial to the correctness of the preprocessor output. Fortunately, this file should change only rarely, such as when a new operator is introduced or (unlikely) an existing one removed.

The contents of the input file consist of a single number on the first line, which represents the number of lines to follow, i.e., the number of operators that will be represented in the table. This number is presently 16. Each subsequent line contains information about a single operator, including the external function name; the internal function name; for each input parameter, the parameter type, the parameter declaration, and an "i" to convey that it is an input parameter; and for the return type, again the type and the declaration, but this time an "o" to identify it as an output parameter.

The type identifies the parameter as being of one of only five possible parameter types, i.e., pixrect, colormap, integer, character string or IMAGE.

The declaration is a string that represents a proper C declaration for that type <u>in that particular internal function</u>. A given parameter type, eg. IMAGE, may have

different declarations from one internal function to another, so this is an essential element of the table. At present only the declarations for the return types, where new declarations must be created, are required. However, for uniformity, the declarations of all input parameters are included in the table as well and must be present in function.isql.

For example, the entry for PIXRECT is as follows:

```
PIXRECT,ISpixrect,image,char *,i,pixrect,struct
pixrect **,o
```

PIXRECT is the external function name, ISpixrect is the corresponding internal function name, the internal function returns a value of type image, the only input parameter has a declaration of char * and its only output has a declaration of struct pixrect **.

"i" and "o" are used to represent the input/output character of the parameter vice some numeric scheme, because each token is read into the table as a character string. This avoids any need for converting back to numeric and preserves the uniformity of the implementation of the table.

Token delimiters are a comma or a new line character (\n), so all other characters are incorporated into tokens. This is necessary to support the inclusion of the declaration, which may contain spaces.

Once loaded, the table takes the form of an array of lists, one item in the array for each of the 16 operators.

Each item is a list of string tokens containing the data described above. This format facilitates comparison with string tokens parsed from the ".isc" file to be preprocessed, and output of declarations to the ".sc" file; no conversions are required.

A function called **load_function_table(<number of entries>)** is executed at the beginning of each preprocessor execution. This loads the contents of **function.isql** into the array structure. The table has a global definition and is available to all portions of the preprocessor. The number of entries (eg. 16) is passed back to the calling program and must be explicitly passed to any function needing that information, such as for scanning the table to determine if an expression represents an ISQL function call.

A second program is available primarily for debugging use in case changes to the **function.isql** file produce anomalous preprocessor results. This function is display_function_table(<number of entries>), and it does just that. The number of entries, generally the value returned from load_function_table, is the only input required. The result is sent to the standard output device, i.e., the CRT.

D. PROCESSING METHOD

The preprocessor reads in the input file one line at a time and examines it to see if it signals the start of an ISQL statement. If it does not, the line is written directly out to the output file with no changes. If it does contain

the beginning of an ISQL statement, it must next determine what type of statement it is, i.e., insert, update, select, etc., and special processing for that particular type of statement begins.

Some statements having no variable parts, such as a BEGIN or END statement, require no special handling, and are simply written to the output file just as a non-ISQL statement. Most, however, require processing to convert one or more parts of the parsed statement from ISQL to ESQL. This special processing varies greatly from one statement to another, but the following general steps apply to all.

First, the incoming statement is parsed and the variable portions, such as column name lists, expression lists, table names, etc., are placed in a tree structure for later retrieval. The preprocessor then examines any portion of the tree which <u>may</u> contain an attribute of type IMAGE or an external function call which returns a value of type IMAGE. For example, a list of expressions is a tree part that would be a candidate for examination.

If an attribute of type IMAGE is encountered, it is replaced in the expression list by its filename/file description pair by adding the _f and _d extensions to the attribute name. In the case of a statement such as an INSERT, where column names are provided to identify the location in which the new data is to be placed, the column

name must be modified in the same manner as the expression, producing two column names in lieu of one.

If an external function call is encountered, it is replaced in the list of expressions by a variable name(s) representing the input parameter(s) or output parameter(s) of the corresponding internal function as appropriate. See the SELECT and INSERT examples respectively in Chapter III, Section E above. To accomplish this, it must first output an ESQL declaration section to uniquely identify the output parameter(s) of the internal function, and it must then output a call to that internal function to load values into those variables.

At this point in preprocessing, the information has been collected to translate the ISQL statement into ESQL equivalent, and the ESQL statement is written to the output file.

Each of these steps is executed repetitively until the entire input file is read, and the output file has been constructed, replacing all ISQL statements by their ESQL equivalents.

E. PROCESSING UTILITIES

In addition to the internal functions included in Table 2, there are two utility functions. One is ISget_names (<image>,<filename>,<description>) which passes back the filename and description identifier pair for the input image identifier.

The second utility function is generate_filename() which generates a unique filename from the system date and time for an image created by ISimage_from_pixrect or ISconstruct_image. This name takes the form

yyjjj.hhmmss

where the first two digits identify the year, the next three digits represent the julian day, and the remaining six digits are the two-digit hour, minute and second, respectively, corresponding to the moment at which the file was created.

F. ERROR HANDLING

Error handling has been held to a minimum. No effort is made to intercept errors which will be evaluated by the INGRES preprocessor or the C compiler. An effort has been made to apprise the user, whenever possible, of the consequences of an error that impacts the proper execution of the ISQL preprocessor.

If a file without the correct ".isc" extension is submitted or if no file name is supplied when the preprocessor is called, the user is notified immediately on the standard output device (CRT) of the error or omission, and the preprocessor is terminated.

Each of the internal functions returns a code upon normal or abnormal termination. Each of these codes is in the range 200 to 299, a group of numbers not already utilized by INGRES. All of these codes are defined in a file called "errors.h." This file also includes codes for messages

intercepted from the pixrect library, encoded in the range 300 to 399. This information should be part of a user's manual so that the user can interpret the result of an operation on an instance of the IMAGE ADT.

V. CONCLUSION

At present, all of the internal functions have been coded. Code for these functions is provided in Appendix A. Testing has been restricted due to the slow process of obtaining the SQL version of INGRES necessary to establish the final link for database image display on the Sun system. All components are now in place and rigorous testing is ongoing.

The state of the preprocessor, the main driving loop for which is supplied in Appendix B, is such that it can load the function table (Appendix C is the pertinent code and Appendix D is a sample of the function table input) and it can iteratively find each occurrence of an EXEC SQL statement. At this time, only the INSERT statement can be fully processed. The code required to process such a statement can be found in Appendix E. All other possible ESQL statements are stubbed to return a message. Appendix F represents a sample input file and Appendix G is the corresponding output file following execution of the preprocessor. A collection of utility functions utilized throughout the preprocessor is assembled in Appendix H and error code definitions are found in Appendix I.

There remain 20 types of statements to implement in the preprocessor to provide full INGRES ESQL functionality for the IMAGE ADT. Several of these types represent a subset of

statement formats in themselves, eg., there are several forms for a SELECT statement. The code unique to the processing of the INSERT statement alone consists of more than 750 lines, so there is still an enormous amount of programming required to attain full implementation.

Following completion of the preprocessor, the logical follow-on is to build a large, practical application that employs it. From such a test, it can be determined if the set of ADT operators that have been provided are appropriate, and functions can be deleted or added as necessary to achieve full ADT utility.

When fully implemented, this preprocessor can be utilized by other higher level applications for processing of databases that incorporate images. These applications will, in most instances, provide yet another layer of abstraction between the database and the end user targeted at providing a "user-friendly" environment for the user seeking information from the database.

The result of related research (Sawyer, 1988) will provide a similar preprocessing capability for incorporation of sound data. A goal following completion of the IMAGE preprocessor will be to incorporate image and sound data in the <u>same</u> database. The ultimate goal is development of a preprocessor that is fully extensible for <u>all</u> data types.

The work already completed has demonstrated the ability to implement an unformatted data type, in this case IMAGE

data, as an ADT. The information content of the ADT has been successfully abstracted from the physical aspects of its storage, hiding such implementation specifics as the fact that the images themselves are stored in separate, standard files or the fact that an image attribute is interpreted for internal use into a filename and a description name. This provides the user with the illusion of a relational database that incorporates the IMAGE data type. The implementation specifics of the ADT can be modified to accommodate changes in the database environment with no modification to the user side of the database interface, an important feature of the use of ADT's.

APPENDIX A

CODE FOR INTERNAL FUNCTIONS

```
/* ************************************************************************** */
#include <stdio.h>
#include <sys/param.h>
#include <sys/types.h>
#include <time.h>
#include <pixrect/pixrect_hs.h>
#include "defines.h"
#include "errors.h"
/* ************************************************************************** */

ISimage_from_pixrect (input_pixrect, colormap, image_filename, image_descr)

/*
**** produce a rasterfile "image_filename" from a pixrect and a colormap
*/

 struct pixrect *input_pixrect;  /* input */
 colormap_t *colormap;           /* input */
 char *image_filename,           /* output */
     *image_descr;               /* output */

{
 FILE *new_file;

/*
**** obtain a unique file name
*/
 generate_filename (image_filename);

 if ((new_file = fopen(image_filename,"w")) == NULL)
    return FOPEN_ERR;

/*
**** create a rasterfile from the pixrect and colormap
*/
 if ((pr_dump (input_pixrect, new_file, colormap, RT_STANDARD, 0)) == PIX_ERR)
   {
    fclose (new_file);
    return PR_DUMP_ERR;
   }

 fclose(new_file);
 return ERROR_FREE;
}
```

```
/* ************************************************************************** */

generate_filename (filename)

char *filename;   /* output */

/*
**** produce a unique filename composed of 2-digit year, julian date,
     hour, minute, and second
*/

{
 char image_filename[13],
     pathname[MAXPATHLEN],
     *p;
 struct tm *t;
 time_t current_time;
 int i;

 current_time = time(NULL);

 t = gmtime(&current_time);

 sprintf(image_filename, "%02d%03d.%02d%02d%02d",
   t->tm_year, t->tm_yday, t->tm_hour, t->tm_min, t->tm_sec);

 image_filename[12]='\0';

 getwd (pathname);

 if (strlen (pathname) <= (FILE_NAME_LENGTH - 14))
   {
    for (i = 0; i < strlen (pathname); i++)
                              *filename++ = pathname[i];
    *filename++ = '/';
   }

 p = image_filename;

 while (*filename++ = *p++)
   ;

 *filename = '\0';

 return;
}
```

```
/* ************************************************************************** */

ISpixrect(filename, file_descr, new_pixrect)

/*
**** extract a pixrect from rasterfile "filename"
*/

 char *filename,  /* input */
     *file_descr;  /* input, not used */
 struct pixrect **new_pixrect;  /* output */

 {
  FILE *input;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((*new_pixrect = pr_load (input, NULL)) == NULL)
    {
                              fclose (input);
     return PR_LOAD_ERR;
    }

  fclose(input);
  return ERROR_FREE;
 }
```

```
/* ************************************************************************ */

ISheight (filename, file_descr, height)

/*
**** extract the image height from rasterfile "filename"
*/

 char *filename,  /* input */
     *file_descr;  /* input, not used */
 int *height;  /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
    {
                                   fclose (input);
     return PR_LOAD_HEADER_ERR;
    }

  fclose(input);
  *height = rh.ras_height;
  return ERROR_FREE;
 }
```

```
/* ************************************************************************** */

ISwidth (filename, file_descr, width)

/*
**** extract the image width from rasterfile "filename"
*/

 char *filename,  /* input */
     *file_descr;   /* input, not used */
 int *width; /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
    {
                                fclose (input);
     return PR_LOAD_HEADER_ERR;
    }

  fclose(input);
  *width = rh.ras_width;
  return ERROR_FREE;
 }
```

```
/* ********************************************************************** */

ISdepth (filename, file_descr, depth)

/*
**** extract the image depth from rasterfile "filename"
*/

 char *filename,  /* input */
     *file_descr;  /* input, not used */
 int *depth;  /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
    {
                                  fclose (input);
     return PR_LOAD_HEADER_ERR;
    }

  fclose(input);
  *depth = rh.ras_depth;
  return ERROR_FREE;
 }
```

```
/* ************************************************************************** */

ISencoding (filename, file_descr, encoding)

/*
**** extract the encoding type from rasterfile "filename"
*/

 char *filename,  /* input */
     *file_descr;  /* input, not used */
 int *encoding;  /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
    {
                    fclose (input);
     return PR_LOAD_HEADER_ERR;
    }

  fclose(input);

/*
**** check the different combinations of rh.ras_type and rh.ras_maptype
*/
  switch (rh.ras_type)
    {
    case RT_STANDARD:
                   switch (rh.ras_maptype)
                    {
                    case RMT_NONE:
                     *encoding = NO_COLORMAP_GREYNESS;
                     break;
                    case RMT_EQUAL_RGB:
                     *encoding = COLORMAP_RGB;
                     break;
                    default:
                     return INVALID_MAP_TYPE_ERR;
                    }
                   break;
    case RT_GREYNESS:
                   switch (rh.ras_maptype)
                    {
                    case RMT_NONE:
                     *encoding = NO_COLORMAP_GREYNESS;
                     break;
```

```
                    case RMT_RAW:
                      *encoding = COLORMAP_GREYNESS;
                      break;
                    default:
                      return INVALID_MAP_TYPE_ERR;
                    }
                  break;
    case RT_RGB:
                  switch (rh.ras_maptype)
                    {
                    case RMT_NONE:
                      *encoding = NO_COLORMAP_RGB;
                      break;
                    case RMT_RAW:
                      *encoding = COLORMAP_RGB;
                      break;
                    default:
                      return INVALID_MAP_TYPE_ERR;
                    }
                  break;
    case RT_IHS:
                  switch (rh.ras_maptype)
                    {
                    case RMT_NONE:
                      *encoding = NO_COLORMAP_IHS;
                      break;
                    case RMT_RAW:
                      *encoding = COLORMAP_IHS;
                      break;
                    default:
                      return INVALID_MAP_TYPE_ERR;
                    }
                  break;
    default:
      return INVALID_RASTER_TYPE_ERR;
    }

  return ERROR_FREE;
}
```

```
/* ************************************************************************ */

IScolormap_length (filename, file_descr, map_length)

/*
**** extract the map length from the rasterfile for "filename"
*/

 char *filename,  /* input */
     *file_descr;  /* input, not used */
 int *map_length;  /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
    {
                         fclose (input);
     return PR_LOAD_HEADER_ERR;
    }

  fclose(input);

  switch (rh.ras_maptype)
    {
    case RMT_NONE:
                         *map_length = 0;
                         break;
    case RMT_EQUAL_RGB:
                         /* this implies that the colormap entry length is 3 bytes */
                         *map_length = rh.ras_maplength / 3;
                         break;
    case RMT_RAW:
                         /* there is no way of determining the original entry length.
                           Therefore, the length in bytes is returned, and a warning
                           is signalled to the calling program */
                         *map_length = rh.ras_maplength;
                         return MAPLENGTH_IN_BYTE_WARNING;
    default:
                         return INVALID_MAP_TYPE_ERR;
    }

  return ERROR_FREE;
 }
```

```
/* ************************************************************************ */

IScolormap_entry_size (filename, file_descr, size)

/*
**** extract the map entry size from rasterfile "filename"
*/

  char *filename,  /* input */
      *file_descr;  /* input, not used */
  int *size;  /* output */

  {
   FILE *input;
   struct rasterfile rh;

/*
**** check to make sure that file exists and is loadable
*/
   if ((input = fopen(filename,"r")) == NULL)
      return FOPEN_ERR;

   if ((pr_load_header(input,&rh)) == PIX_ERR)
     {
                         fclose (input);
      return PR_LOAD_HEADER_ERR;
     }

   fclose(input);
/*
**** a fixed constant for SUN is returned
*/
   switch (rh.ras_type)
     {
     case RT_OLD:
     case RT_STANDARD:
                         *size = 3;   /* 24 bits/3 bytes for Sun colormaps */
                         break;
     default:
                         return NO_COLORMAP_ENTRY_SIZE_ERR;
     }

   return ERROR_FREE;
  }
```

```
/* ******************************************************************************** */

IScolormap (filename, file_descr, cm_ptr)

/*
**** extract the colormap from rasterfile "filename"
*/

 char *filename, /* input */
     *file_descr; /* input, not used */
 colormap_t *cm_ptr; /* output */

 {
  FILE *input;
  struct rasterfile rh;

  if ((input = fopen(filename,"r")) == NULL)
    return FOPEN_ERR;

  if ((pr_load_header(input,&rh)) == PIX_ERR)
   {
                        fclose (input);
    return PR_LOAD_HEADER_ERR;
   }

  switch (rh.ras_maptype)
   {
   case RMT_NONE:
    cm_ptr->type = RMT_NONE;
                        cm_ptr->length = 0;
                        cm_ptr->map[0] = NULL;
                        cm_ptr->map[1] = NULL;
                        cm_ptr->map[2] = NULL;
                        break;
   case RMT_EQUAL_RGB:
                        if (pr_load_colormap (input, &rh, cm_ptr) == PIX_ERR)
                         {
                          fclose (input);
                          return PR_LOAD_COLORMAP_ERR;
                         }
                        break;
   case RMT_RAW:
                        if (pr_load_colormap (input, &rh, cm_ptr) == PIX_ERR)
                         {
                          fclose (input);
                          return PR_LOAD_COLORMAP_ERR;
                         }
                        /* in case of RMT_RAW the function places the address of the
                          colormap in cm_ptr->map[0]. The other two pointers are not
                          not NULL, but contain nonsense that can cause segmentation
                          fault. Therefore they are set to NULL: */
```

```
                    cm_ptr->map[1] = NULL;
                    cm_ptr->map[2] = NULL;
                    break;
  default:
                    fclose (input);
                    return INVALID_MAP_TYPE_ERR;
  }

 fclose(input);
 return ERROR_FREE;
}
```

```
/* ************************************************************************** */

ISpixelmatrix (filename, file_descr, matrixptr)

/*
**** extract the pixel matrix for the image from rasterfile "filename"
*/

 char *filename, /* input */
     *file_descr; /* input, not used */
 unsigned char *matrixptr; /* output */

 {
  FILE *input;
  struct rasterfile rh;
  struct pixrect *pr;
  struct mpr_data *data_ptr;
  long int length,
        i,
        j;
  unsigned char *source,
             *malloc ();
  int out_bytes_per_line,
     in_bytes_per_line;

  if ((input = fopen(filename,"r")) == NULL)
     return FOPEN_ERR;

  if (pr_load_header (input, &rh) == PIX_ERR)
   {
                              fclose (input);
                              return PR_LOAD_HEADER_ERR;
   }

  /* skip over colormap, if any */
  if (rh.ras_maptype != RMT_NONE)
   if (pr_load_colormap (input, &rh, NULL) == PIX_ERR)
                              {
                               fclose (input);
                               return PR_LOAD_COLORMAP_ERR;
                              }

  if (rh.ras_type == RT_STANDARD)
   {
                              if ((pr = pr_load_std_image (input, &rh, NULL)) == NULL)
                               {
                                fclose (input);
                                return PR_LOAD_ERR;
                               }
                              data_ptr = (struct mpr_data *) pr->pr_data;
                              source = (unsigned char *) data_ptr->md_image;
```

```
   }
  else
   /* if rh.ras_type is not RT_STANDARD, pr_load_image calls a
                              decompression algorithm. It must be located in a file whose
                              name is derived from the value of rh.ras_type. Since we don't
                              use those decompression techniques, we must load the pixels
                              ourselves. */
   {
                             source = malloc (rh.ras_length);
                             if (source == NULL)
                              {
                               fclose (input);
                               return MALLOC_ERR;
                              }
                             if (fread (source, sizeof (char), rh.ras_length, input)
                               < rh.ras_length)
                              {
                               fclose (input);
                               return FREAD_ERR;
                              }
   }

  fclose(input);

/* Now source points to the pixelmatrix in the pixrect format, i.e.
  lines are a multiple of 16 bits. To comply with arbitrary pixel-
  matrix formats, the trailing byte (if existing) is stripped off. */

/*
**** retrieve the pixrect data
*/
  out_bytes_per_line = (rh.ras_width * rh.ras_depth - 1) / 8 + 1;
  in_bytes_per_line = ((rh.ras_width * rh.ras_depth - 1) / 16 + 1) * 2;

  if (out_bytes_per_line == in_bytes_per_line)
   for (i = 0; i < rh.ras_length; i++)
                             *matrixptr++ = *source++;
  else
   /* out_bytes_per_line == in_bytes_per_line - 1 */
   for (i = 0; i < rh.ras_height; i++)
                             {
                              for (j = 0; j < out_bytes_per_line; j++)
                               *matrixptr++ = *source++;
                              source++;   /* skip filler byte at end */
                             }

  return ERROR_FREE;
 }
```

```
/* ************************************************************************** */

ISwindow (infilename, infile_descr, x, y, dx, dy, outfilename, outfile_descr)

/*
**** extract a sub-image from an existing image
*/

 char *infilename,      /* input */
     *infile_descr,   /* input, not used */
     *outfilename,    /* output */
     *outfile_descr;  /* output */
 int x,
    y,
    dx,
    dy;     /* input */

 {
  FILE *input,
      *output;
  struct rasterfile rh;
  colormap_t cm;
  struct pixrect *input_pr,
             *output_pr;
  struct mpr_data *data_ptr;
  unsigned char *matrix_ptr,
            *target,
            *subimage,
            *malloc ();
  int out_bytes_per_line,
     i,
     j;

  /* The first pixel of the image in the northwest corner has the
    "coordinates" (0, 0). */
  if (x < 0 || y < 0 || dx < 1 || dy < 1)
   {
                          return INVALID_WINDOW_PARAMS;
   }

  if ((input = fopen(infilename,"r")) == NULL)
   {
    return FOPEN_ERR;
   }

  /* reading the image is done almost in the same way as in
    ISpixelmatrix.c. */

  if ((pr_load_header (input, &rh)) == PIX_ERR)
   {
                          fclose (input);
```

```
  return PR_LOAD_HEADER_ERR;
 }

/* The last pixel of the image in the southeast corner has the
  "coordinates" (rh.ras_width - 1, rh.ras_height - 1). */
if (x >= rh.ras_width || y >= rh.ras_height)
 {
                  fclose (input);
                  return NO_WINDOW_OVERLAP;
 }

switch (rh.ras_maptype)
 {
 case RMT_NONE:
  cm.type = RMT_NONE;
                  cm.length = 0;
                  cm.map[0] = NULL;
                  cm.map[1] = NULL;
                  cm.map[2] = NULL;
                  break;
 case RMT_EQUAL_RGB:
                  if (pr_load_colormap (input, &rh, &cm) == PIX_ERR)
                   {
                    fclose (input);
                    return PR_LOAD_COLORMAP_ERR;
                   }
                  break;
 case RMT_RAW:
                  if (pr_load_colormap (input, &rh, &cm) == PIX_ERR)
                   {
                    fclose (input);
                    return PR_LOAD_COLORMAP_ERR;
                   }
                  /* in case of RMT_RAW the function places the address of the
                    colormap in cm_ptr->map[0]. The other two pointers are not
                    not NULL, but contain nonsense that can cause segmentation
                    fault. Therefore they are set to NULL: */
                  cm.map[1] = NULL;
                  cm.map[2] = NULL;
                  break;
 default:
                  fclose (input);
                  return INVALID_MAP_TYPE_ERR;
 }

if (rh.ras_type == RT_STANDARD)
 {
                  if ((input_pr = pr_load_std_image (input, &rh, NULL)) == NULL)
                   {
                    fclose (input);
                    return PR_LOAD_ERR;
```

```
                        }
}
else
 /* if rh.ras_type is not RT_STANDARD, pr_load_image calls a
                        decompression algorithm. It must be located in a file whose
                        name is derived from the value of rh.ras_type. Since we don't
                        use those decompression techniques, we must load the pixels
                        ourselves. */
 {
                        if ((input_pr = mem_create (rh.ras_width, rh.ras_height,
                                rh.ras_depth)) == NULL)
                         {
                          fclose (input);
                          return MEM_CREATE_ERR;
                         }

                        data_ptr = (struct mpr_data *) input_pr->pr_data;
                        matrix_ptr = (unsigned char *) data_ptr->md_image;

                        if (fread (matrix_ptr, sizeof (char), rh.ras_length, input)
                           < rh.ras_length)
                         {
                          fclose (input);
                          return FREAD_ERR;
                         }
 }

fclose(input);

generate_filename (outfilename);

if ((output = fopen (outfilename, "w")) == NULL)
 {
                        return FOPEN_ERR;
 }

/* pr_region builds a secondary pixrect with its own struct
  pixrect and its own struct mpr_data, but referencing the
  same pixelmatrix (pointer md_image in mpr_data). The offset
  information in the mpr_data structure tells procedures like
  pr_dump which part of the pixelmatrix must be used.
  If dx and dy extend beyond the image in input_pr, pr_region
  will reduce them. The real dx and dy values will then be stored
  in outpur_pr->pr_size.x and output_pr->pr_size.y. */

if ((output_pr = pr_region (input_pr, x, y, dx, dy)) == NULL)
 {
                        return PR_REGION_ERR;
 }

if (rh.ras_type == RT_STANDARD && rh.ras_depth == 8)
```

```
{
                    /* pr_dump will set rh.ras_type to RT_STANDARD anyway, so any
                      other value would simply be lost. pr_region works with
                      a depth other than 8, but pr_dump would not write such
                      a secondary pixrect. It would not return an error, either. */

                    if ((pr_dump (output_pr, output, &cm, RT_STANDARD, 0)) != 0)
                      {
                       fclose (output);
                       return PR_DUMP_ERR;
                      }
}
else
{
                    if (rh.ras_depth != 8)
                      {
                       /* this implies actually copying the pixelmatrix on a
                          bit-by-bit basis! We postpone it. */
                       fclose (output);
                       return WINDOW_WRONG_DEPTH_ERR;
                      }

                    /* the following sequence is a simplified version of
                      ISconstruct_image.c, see the comments there. */

                    rh.ras_width = output_pr->pr_size.x;
                    rh.ras_height = output_pr->pr_size.y;
                    /* they can be different from the original dx and dy, if dx and dy
                      were too large. */

                    out_bytes_per_line = ((rh.ras_width - 1) / 2 + 1) * 2;
                    /* this only works because rh.ras_depth is 8! */
                    rh.ras_length = rh.ras_height * out_bytes_per_line;

                    /* all the other components of rh remain as the are, and so
                      does the colormap in cm. */

                    if (pr_dump_header (output, &rh, &cm) == PIX_ERR)
                      {
                       fclose (output);
                       return PR_DUMP_HEADER_ERR;
                      }

                    data_ptr = (struct mpr_data *) output_pr->pr_data;

                    /* copy the subimage to different storage area: */
                    target = malloc (rh.ras_length);
                    subimage = target;   /* save base address */

                    for (i = 0; i < rh.ras_height; i++)
                      {
```

```
                matrix_ptr = (unsigned char *) data_ptr->md_image
                        + (y + i) * data_ptr->md_linebytes   /* skip lines */
                        + x;
                for (j = 0; j < rh.ras_width; j++)
                  *target++ = *matrix_ptr++;
                if (rh.ras_width == (out_bytes_per_line - 1))
                  *target++ = '\0';
              }

            if (fwrite (subimage, sizeof (unsigned char), rh.ras_length,
                        output)
                < rh.ras_length)
              {
                fclose (output);
                free (subimage);
                return FWRITE_ERR;
              }
    }

  fclose (output);
  free (subimage);
  pr_destroy (output_pr);
  pr_destroy (input_pr);

  *outfile_descr = '\0';

  return ERROR_FREE;
}
```

```
/* ************************************************************************ */

ISadd_description(infilename, indescr, newdescr, outfilename, outdescr)

/*
**** add to the description of an image
*/

 char *infilename,     /* input */
     *indescr,       /* input */
     *newdescr,      /* input */
     *outfilename,   /* output */
     *outdescr;      /* output */

 {
  int i = 0;

  while (*outfilename++ = *infilename++)
   ;

  while (*outdescr++ = *indescr++)
   i++;

  outdescr--;  /* reposition on '\0' */

  while ((*outdescr++ = *newdescr++) && i < MAX_DESCR)
   i++;

  if (i == MAX_DESCR && *outdescr != '\0')
   {
    *outdescr = '\0';
                  return DESCR_TOO_LONG_ERR;
   }

  return ERROR_FREE;
 }
```

```
/* ************************************************************************ */

ISreplace_description(infilename, indescr, newdescr, outfilename, outdescr)

/*
**** replace the description of an image
*/

 char *infilename,       /* input */
     *indescr,         /* input */
     *newdescr,        /* input */
     *outfilename,     /* output */
     *outdescr;        /* output */

 {
  while (*outfilename++ = *infilename++)
    ;
  while (*outdescr++ = *newdescr++)
    ;
  return ERROR_FREE;
 }
```

```
/* ***************************************************************************** */

ISdescription_length (filename, descr, char_count)

/*
**** count the characters in the description of an image
*/

 char *filename;          /* input, not used */
 char *descr;             /* input */
 unsigned int *char_count; /* output */

 {
  unsigned int i = 0;

  while (*descr++ != '\0')
   i++;

  *char_count = i;
  return ERROR_FREE;
 }
```

```
/* ************************************************************************* */

ISdescription (filename, old_descr_name, new_descr_name)

/*
**** copy the description to the output
*/

 char *filename,    /* input, not used */
     *old_descr_name,   /* input */
     *new_descr_name;   /* output */

 {
  while (*new_descr_name++ = *old_descr_name++)
   ;
  return ERROR_FREE;
 }
```

```
/* ************************************************************************** */

ISshows (filename, descr, pattern, match)

/*
**** determine whether or not string "pattern" is contained
**** within string "descr"; returns 1 if true, 0 if false
*/

 char *filename,   /* input, not used */
      *descr,      /* input */
      *pattern;    /* input */
 int  *match;      /* output */

 {
  int i,
      j,
      found;

  if (*pattern == '\0')
    found = 1;  /* NULL string always is contained */
  else
    found = 0;  /* initialize found for loop use */

  i = 0;

  while (*(descr+i) != '\0' && !found)
    {
     if (*(descr+i) == *pattern)
       {
        j = 0;
        while (*(descr+i+j) == *(pattern+j) && *(pattern+j) != '\0')
          j++;
        if (*(pattern+j) == '\0')  /* pattern matched */
          found = 1;
        else
          if (*(descr+i+j) == '\0')   /* pattern longer than
                                         remaining descr */
            i = i + j;  /* terminate outer loop */
          else  /* continue search starting with next letter in descr */
            i++;
       }
     else
       i++;
    }

  *match = found;
  return ERROR_FREE;
 }
```

```
/* ****************************************************************************** */

ISconstruct_image (width, height, depth, encoding, colormap_length,
                   colormap_entry_length, colormap, pixelmatrix,
                   image_filename,image_descr)

/*
**** construct an image located in file "filename" from the
**** discrete parts of the pixrect and the colormap
*/

int width,
    height,
    depth,
    encoding,
    colormap_length,
    colormap_entry_length;
unsigned char *pixelmatrix,
              *colormap;
char *image_filename,
     *image_descr;

{
 int in_bytes_per_line,
     out_bytes_per_line;
 int i,
     j;
 unsigned char *target;
 struct pixrect *image;
 struct rasterfile rh;
 struct mpr_data *data_ptr;
 colormap_t cm;
 unsigned char red[256],
               green[256],
               blue[256];
 FILE *new_file;

 if (width <= 0)
  return NO_WIDTH_ERR;

 if (height <= 0)
  return NO_HEIGHT_ERR;

 if (depth <= 0)
  return NO_DEPTH_ERR;

 if (encoding < 0 || encoding > 6)
  return INVALID_ENCODING_ERR;

/*
**** assemble the rasterfile header
```

```
*/
 rh.ras_magic = RAS_MAGIC;
 rh.ras_width = width;
 rh.ras_height = height;
 rh.ras_depth = depth;

 out_bytes_per_line = ((width * depth - 1) / 16 + 1) * 2;
 /* lines must be rounded up to multiples of 16-bit words */
 rh.ras_length = height * out_bytes_per_line;
 /* what if we have more than 32768 bytes? big problem */

 switch (encoding)
  {
  case NO_COLORMAP_GREYNESS:
   rh.ras_type = RT_STANDARD;
   rh.ras_maptype = RMT_NONE;
   rh.ras_maplength = 0;
   cm.type = RMT_NONE;
   cm.length = 0;
   break;
  case NO_COLORMAP_RGB:
   rh.ras_type = RT_RGB;
   rh.ras_maptype = RMT_NONE;
   rh.ras_maplength = 0;
   cm.type = RMT_NONE;
   cm.length = 0;
   break;
  case NO_COLORMAP_IHS:
   rh.ras_type = RT_IHS;
   rh.ras_maptype = RMT_NONE;
   rh.ras_maplength = 0;
   cm.type = RMT_NONE;
   cm.length = 0;
   break;
  case COLORMAP_GREYNESS:
   rh.ras_type = RT_GREYNESS;
   rh.ras_maptype = RMT_RAW;
   rh.ras_maplength = colormap_entry_length * colormap_length;
   cm.type = RMT_RAW;
   cm.length = rh.ras_maplength;
   cm.map[0] = colormap;
   cm.map[1] = NULL;
   cm.map[2] = NULL;
   break;
  case COLORMAP_RGB:
   rh.ras_maplength = colormap_entry_length * colormap_length;
   if (colormap_entry_length == 3 && colormap_length <= 256)
                    {
                     rh.ras_type = RT_STANDARD;
                     rh.ras_maptype = RMT_EQUAL_RGB;
                     for (i = 0; i < colormap_length; i++)
```

```
                        {
                         red[i] = *colormap++;
                         green[i] = *colormap++;
                         blue[i] = *colormap++;
                        }
                       cm.type = RMT_EQUAL_RGB;
                       cm.length = colormap_length;
                       /* it took hours to find this. pr_dump_header verifies that
                         rh.ras_maplength is three times cm.length, if cm.type ==
                         RMT_EQUAL_RGB. In contrast to that cm.length must be equal
                         to rh.ras_maplength in case of RMT_RAW. */
                       cm.map[0] = red;
                       cm.map[1] = green;
                       cm.map[2] = blue;
                      }
  else
                      {
                       rh.ras_type = RT_RGB;
                       rh.ras_maptype = RMT_RAW;
                       cm.type = RMT_RAW;
                       cm.length = rh.ras_maplength;
                       cm.map[0] = colormap;
                       cm.map[1] = NULL;
                       cm.map[2] = NULL;
                      }
  break;
 case COLORMAP_IHS:
  rh.ras_type = RT_IHS;
  rh.ras_maptype = RMT_RAW;
  rh.ras_maplength = colormap_entry_length * colormap_length;
  cm.type = RMT_RAW;
  cm.length = rh.ras_maplength;
  cm.map[0] = colormap;
  cm.map[1] = NULL;
  cm.map[2] = NULL;
  break;
 }

/*
**** open a new file with a unique file name
*/
 generate_filename(image_filename);

 if ((new_file = fopen(image_filename,"w")) == NULL)
    return FOPEN_ERR;

/*
**** dump the header information and the colormap into the file
*/
 /* if rh.ras_maptype contains RMT_RAW, pr_dump_header will only use
   cm.map[0]. This pointer must point to a storage area that contains
```

```
    the colormap in the length specified in rh.maplength and cm.length */

  if (pr_dump_header (new_file, &rh, &cm) == PIX_ERR)
    {
     fclose (new_file);
     return PR_DUMP_HEADER_ERR;
    }

/*
**** create a memory pixrect
*/
  if ((image = mem_create (width, height, depth)) == NULL)
    {
     fclose (new_file);
     return MEM_CREATE_ERR;
    }

/*
**** initialize the memory pixrect
*/
  data_ptr = (struct mpr_data *) image->pr_data;
  target = (unsigned char *) data_ptr->md_image;

  in_bytes_per_line = (width * depth - 1) / 8 + 1;

  if (in_bytes_per_line == out_bytes_per_line)
    /* no need to copy, pixelmatrix is in proper format */
    data_ptr->md_image = (short *) pixelmatrix;
  else
    /* in_bytes_per_line == (out_bytes_per_line - 1) */
    for (i = 0; i < height; i++)
      {
       for (j = 0; j < in_bytes_per_line; j++)
         *(target+(i*out_bytes_per_line)+j) = *pixelmatrix++;
                              *(target+(i*out_bytes_per_line)+in_bytes_per_line) =
                               (unsigned char) 0;
      }

/*
**** dump the image into the file
*/
  if (rh.ras_type == RT_STANDARD)
    {
     if (pr_dump_image (image, new_file, &rh) == PIX_ERR)
                              {
                               fclose (new_file);
                               return PR_DUMP_IMAGE_ERR;
                              }
    }
  else
    /* if rh.ras_type is not RT_STANDARD, pr_dump_image will call a
```

```
    compaction program that must be available in some file.
    Since we don't have that and don't want compaction either,
    we must write the image without the help of pr_dump_image */

  if (fwrite ((unsigned char *) data_ptr->md_image,
                                sizeof (unsigned char), rh.ras_length,
                                new_file)
                          < rh.ras_length)
    {
                          fclose (new_file);
                          return FWRITE_ERR;
    }

 *image_descr = '\0';

 fclose(new_file);
 return ERROR_FREE;
}
```

APPENDIX B

MAIN PROGRAM CODE

```
#define TABLE_DATA_FILE "function.isql"

/* length of the two representation attributes filename and description */
#define FILE_NAME_LENGTH 64
#define MAX_DESCR 500

/* values for encoding */
#define COLORMAP_RGB 1
#define COLORMAP_IHS 2
#define COLORMAP_GREYNESS 3
#define NO_COLORMAP_RGB 4
#define NO_COLORMAP_IHS 5
#define NO_COLORMAP_GREYNESS 6

/* additional values for the ras_type field in the rasterfile header */
#define RT_GREYNESS 101
#define RT_RGB 102
#define RT_IHS 103

#define MAX_LINE 256
#define MAX_TOKEN 256
#define MAX_TABLENAME 12
#define MAX_FUNC_NAME 25
#define MAX_SQL_NAME 10
#define MAX_PARAMS 15
#define ADD_INDENT 4

#define TRUE 1
#define FALSE 0

#define SPACE ' '
#define TAB '\t'
#define NEW_LINE '\n'
#define LEFT_PAREN '('
#define RIGHT_PAREN ')'
#define COMMA ','
#define SEMI_COLON ';'
#define COLON ':'
#define UNDERSCORE '_'
#define TERMINAL '\0'
#define PERIOD '.'
#define LEFT_BRACKET '{'
#define RIGHT_BRACKET '}'
```

```
#define VALUES_CLAUSE 2
#define SUBQUERY 3

#define ABORT_OP 20
#define BEGIN_OP 21
#define CLOSE_OP 22
#define COPY_OP 23
#define CONNECT_OP 24
#define CREATE_OP 25
#define DECLARE_OP 26
#define DELETE_OP 27
#define DISCONNECT_OP 28
#define DROP_OP 29
#define END_OP 30
#define FETCH_OP 31
#define HELD_OP 32
#define INCLUDE_OP 33
#define INSERT_OP 34
#define MODIFY_OP 35
#define OPEN_OP 36
#define PRINT_OP 37
#define RELOCATE_OP 38
#define SAVE_OP 39
#define SAVEPOINT_OP 40
#define SELECT_OP 41
#define SET_OP 42
#define UPDATE_OP 43
#define WHENEVER_OP 44



/* ************************************************************************* */

/*
  The table of functions, their translations, and their parameters is con-
  structed as an array of nodes as defined here.
*/

struct func_tab_entry
 {
  char *func_name,
      *ISfunc_name;
  struct param *in_param_list,
           *return_type;
 };

struct func_tab_entry **func_tab;  /* global declaration of function table */

/* ************************************************************************* */
```

```
/*
   Structure for constructing a list of parameters.
*/

struct param
  {
   char *param_type,
        *param_decl;
   struct param *next;
  };

/* ************************************************************************ */

/*
   Structure for saving the variable syntactic parts of an insert
   statement.
*/

struct insert_node_struct
  {
   char *tablename;
   struct column *col_list;
   struct expr *expr_list;
  };

/* ************************************************************************ */

/*
   Structure for constructing a list of column names.
*/

struct column
  {
   char *column_name;
   struct column *next;
  };

/* ************************************************************************ */

/*
   Structure for constructing a list of expressions.
*/

struct expr
  {
   char *expression;
   struct expr *next;
  };

/* ************************************************************************ */
```

```
/*
*/

struct ISfunc_struct
  {
   char *name;
   struct ISparam *param_list;
  };

/* ****************************************************************************** */

/*
   Structure for constructing a list of IS function parameters.
*/

struct ISparam
  {
   char *name;
   struct ISparam *next;
  };
```

```
/* ************************************************************************** */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "errors.h"
#include "defines.h"
#include "structures.h"

/* ************************************************************************** */

main(argc,argv)    /* argv contains name of program to be preprocessed */

/*
   Preprocessor main program.  Reads in name of ISQL file to be pre-
   processed, verifies it, loads function table, and initiates
   convertion of ISQL statements to ESQL equivalents.  Input is
   ".isc" file, output is ".sc" file.
*/

 int argc;
 char *argv[];

 {
  FILE *input,
      *output;
  char *err_code,
      *operation,
      *argument1,
      input_filename[MAX_TOKEN + 1],
      input_filename_ext[MAX_TOKEN + 1],
      *output_filename,
      inbuff[MAX_LINE + 1];
  int  i,
      j,
      num_entries;

  /* check for presence of filename argument */

  if (argc!=2)
    {
     printf("\n\n>>>>> Please enter name of program to be preprocessed.\n\n");
     exit();
    }

  /* ensure that filename ends in ".isc" */

  argument1 = argv[1];

  while (((input_filename[i] = argument1[i]) != TERMINAL) &&
        ((input_filename[i] = argument1[i]) != PERIOD))
```

```
  i++;
 input_filename[i] = TERMINAL;

 if (argument1[i] != PERIOD)
   {
    printf("\n\n>>>>> Input filename must have an extension \".isc\"\n\n");
    exit();
   }
 else
   {
    i++;
    while (input_filename_ext[j] = argument1[i])
      {
       i++;
       j++;
      }
    input_filename_ext[j+1] = TERMINAL;
    if (strcmp(input_filename_ext,"isc") != 0)
      {
       printf("\n\n>>>>> Input filename must have an extension \".isc\"\n\n");
       exit();
      }
    else
     printf ("\n\n>>>>> Preprocessing file %s.%s\n",input_filename, input_filename_ext);
   }

 output_filename = strcat(input_filename,".sc");

 num_entries = load_function_table();

 /* open input file */

 if ((input = fopen (argv[1],"r")) == NULL)
   return FOPEN_ERR;

 /* open output file */

 if ((output = fopen (output_filename,"w")) == NULL)
   return FOPEN_ERR;

 /* find and evaluate EXEC SQL statements */

 while (get_next_sql(input,output,inbuff,&operation))
   process_next_sql(input,output,inbuff,operation,num_entries);

 fclose(input);
 fclose(output);

 printf ("\n>>>>> Preprocessor terminated.\n\n");
}
```

```
#include <stdio.h>
#include <string.h>
#include "defines.h"

/* ************************************************************************ */

get_next_sql(input_file,output_file,inbuff,operation)

/*
  Reads a line from the input file.  If it is an EXEC SQL statement, the line
  is returned to the main program for processing.  Otherwise, the line is
  written unchanged to the output file.  Called repetitively in while loop
  by main program.
*/

 FILE *input_file,
     *output_file;
 char inbuff[MAX_LINE + 1],
     **operation;

 {
  char token1[MAX_TOKEN + 1],
      token2[MAX_TOKEN + 1],
      *err_code;
  static char token3[MAX_TOKEN + 1],
            token4[MAX_TOKEN + 1];
  int i,
     inbuff_ndx;

  while (err_code = fgets (inbuff,MAX_LINE,input_file))
   {
    inbuff_ndx = 0;

    /*  skip the white space at the beginning of the line */
    while ((inbuff[inbuff_ndx] == SPACE) ||
         (inbuff[inbuff_ndx] == TAB))
     inbuff_ndx++;

    /*  pick off each of the first four tokens for use in determining
       if this is an EXEC SQL statement  */

    get_next_token(inbuff,inbuff_ndx,token1,&inbuff_ndx);
    to_upper_case(token1);

    get_next_token(inbuff,inbuff_ndx,token2,&inbuff_ndx);
    to_upper_case(token2);

    get_next_token(inbuff,inbuff_ndx,token3,&inbuff_ndx);
    to_upper_case(token3);

    get_next_token(inbuff,inbuff_ndx,token4,&inbuff_ndx);
```

```
    to_upper_case(token4);

    if ((strcmp(token1,"EXEC") == 0) && (strcmp(token2,"SQL") == 0))
      {
       /* EXEC SQL statement without a label */
       *operation = token3;
       return TRUE;
      }
    else if ((strcmp(token2,"EXEC") == 0) && (strcmp(token3,"SQL") == 0))
      {
       /* there is a label preceding the EXEC SQL statement */
       *operation = token4;
       return TRUE;
      }
    else
     /* this is not an EXEC SQL statement, write the line to the
        output file */
     fputs (inbuff,output_file);
  }
}
```

```
#include <stdio.h>
#include "defines.h"

/* ****************************************************************************** */

process_next_sql(input,output,inbuff,operation,num_entries)

/*
   Accepts a line of text in inbuff that represents the first (maybe only)
   line of an EXEC SQL statement. The type of statement is identified by
   operation. Uses a switch statement to peform the appropriate pro-
   cessing, reading additional input lines as necessary. Writes appropriate
   corresponding lines of code to output. Called repetitively from main
   program whenever an EXEC SQL statement is encountered.
*/

 FILE *input,
     *output;
 char inbuff[MAX_LINE + 1],
     *operation;
 int num_entries;

 {
  char outbuff[MAX_LINE + 1];
  int i,
      inbuff_ndx,
      outbuff_ndx;

  /* Intitialize the output buffer. */
  for (i=0; i<MAX_LINE+1; i++)
    outbuff[i] = TERMINAL;

  inbuff_ndx = 0;
  outbuff_ndx = 0;

  /* all operations evaluated for type in upper case */
  to_upper_case(operation);

  switch(get_operation_type(operation)) {
   case INSERT_OP:
      process_insert(input,output,inbuff,num_entries);
      break;
   case BEGIN_OP:
   case DISCONNECT_OP:
   case END_OP:
   case INCLUDE_OP:
      /* Move the contents of the input buffer to the output
         buffer, up to but not including the terminating
         null character. */
      while ((outbuff[outbuff_ndx] = inbuff[inbuff_ndx]) != TERMINAL)
        {
```

```
            inbuff_ndx++;
            outbuff_ndx++;
           }
        break;
      default:
         printf ("\nWarning: sql %s statement not checked for\
image attributes.\n",operation);

         /* Move the contents of the input buffer to the output
            buffer, up to but not including the terminating
            null character. */
         while ((outbuff[outbuff_ndx] = inbuff[inbuff_ndx]) != TERMINAL)
          {
            inbuff_ndx++;
            outbuff_ndx++;
          }
      }

   /* Place the terminating null character at the end of the output
      buffer. */

   outbuff[outbuff_ndx+1] = TERMINAL;

   /* write the output buffer to the output file */
   fputs (outbuff,output);
  }
```

APPENDIX C

CODE FOR PROCESSING FUNCTION TABLE

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include "defines.h"
#include "structures.h"
#include "errors.h"

/* *************************************************************************** */

load_function_table()

/*
   Takes the contents of the data file which contains the information
   needed for translation of an IMAGE function into an ISimage func-
   tion and places it in a table.  This table takes the form of a
   list of lists.  Called once by main program as preprocessing begins.
*/

 {
   struct func_tab_entry *curr_node;
   struct param *curr_in_param,
             *next_param;
   FILE *input;
   int ent_cnt,
      first_in_param,
      list_done,
      i;
   char *token,
       *param_type,
       *param_decl,
       *param_i_or_o,
       buff[MAX_LINE + 1];

   if ((input = fopen (TABLE_DATA_FILE,"r")) == NULL)
      return FOPEN_ERR;

   /* The first item of information, on a line by itself, is the number
      of items (lists) contained in the table.  This is captured and used
      to determine the number of lists in the array and as the counter
      boundary for the following "for" loop.  The new line character is
      discarded. */

   fscanf (input, "%d%*c",&ent_cnt);
```

```
/* Create the initial array of empty lists, using the number of entries
   to determine the size. */
func_tab =
  (struct func_tab_entry **)malloc(sizeof(struct func_tab_entry *) * (ent_cnt + 1));

for (i=0; i<ent_cnt; i++)
  {
   list_done = FALSE;
   first_in_param = TRUE;

   fgets (buff,MAX_LINE,input);  /* get line of data from file */

   token = strtok (buff,",\n");  /* get the first token in the line */

   if ((token == NULL) && (i<(ent_cnt-1)))
                 /* number of entries does not match length of file */
      return FUNC_TAB_ERR;
   else
     {
      /* create a new node for a function's data */
      curr_node = (struct func_tab_entry *) malloc (sizeof(struct func_tab_entry));

      /* allocate space for the function name */
      curr_node->func_name = malloc (strlen(token)+1);
      /* load the function name into the node */
      strcpy(curr_node->func_name,token);

      /* get the ISfunction name */
      token = strtok(NULL,",\n");
      /* allocate space for the ISfunction name */
      curr_node->ISfunc_name = malloc (strlen(token)+1);
      /* load the ISfunction name into the node */
      strcpy(curr_node->ISfunc_name,token);

      curr_node->in_param_list = NULL;
      curr_node->return_type = NULL;

      func_tab[i] = curr_node;

      while (!list_done)
        {
         param_type = strtok(NULL,",\n");
         if (param_type == NULL) /* no more parameters */
           {
            list_done = TRUE;
            continue;
           }
         param_decl = strtok(NULL,",\n");
         if (param_decl == NULL) /* parameter without i or o */
           {
            return PARAM_ERR;
```

```
          }
        param_i_or_o = strtok(NULL,",\n");
        if (param_i_or_o == NULL) /* parameter without i or o */
          {
           return PARAM_ERR;
          }

           /* allocate space for a parameter node */
           next_param = (struct param *) malloc (sizeof(struct param));

           /* allocate space for the parameter type */
           next_param->param_type = malloc(strlen(param_type)+1);
           /* load the parameter declaration */
           strcpy(next_param->param_type,param_type);

           /* allocate space for the parameter decl */
           next_param->param_decl = malloc(strlen(param_decl)+1);
           /* load the parameter declaration */
           strcpy(next_param->param_decl,param_decl);

           next_param->next = NULL;

        if (strcmp(param_i_or_o,"i") == 0)
          {
         /* add an input parameter */
           if (first_in_param)
             {
              first_in_param = FALSE;
              curr_node->in_param_list = next_param;
             }
           else
              curr_in_param->next = next_param;
           curr_in_param = next_param;
          }
        else if (strcmp(param_i_or_o,"o") == 0)
         /* add an output parameter */
           curr_node->return_type = next_param;
        else
         /* error */
          {
           return PARAM_ERR;
          }
        } /* list_done while */

    } /* else */
  } /* while */

 fclose(input);

 return ent_cnt;
}
```

```
#include <stdio.h>
#include "structures.h"

display_function_table(num_entries)

/*
  Utility program for printing contents of function table. Input parameter
  is number of elements in the table array. Not executed as part of
  normal processing.
*/

 int num_entries;

{
 int i;
 struct param *next;

 for (i=0; i<num_entries; i++)
   {
    printf("\nfunction name = %s\n",func_tab[i]->func_name);
    printf("ISfunction name = %s\n",func_tab[i]->ISfunc_name);

    next = func_tab[i]->in_param_list;
    while (next != NULL)
      {
       printf("input param type = %s",next->param_type);
       printf("       decl = %s\n",next->param_decl);
       next = next->next;
      }

    next = func_tab[i]->return_type;
    printf("return type = %s",next->param_type);
    printf("       decl = %s\n",next->param_decl);
   }
}
```

APPENDIX D

SAMPLE FUNCTION TABLE INPUT FILE

```
16
IMAGE_FROM_PIXRECT,ISimage_from_pixrect,pixrect,struct pixrect *,i,colormap,
colormap_t *,i,image,char ,o
PIXRECT,ISpixrect,image,char *,i,pixrect,struct pixrect **,o
HEIGHT,ISheight,image,char *,i,int,int *,o
DEPTH,ISdepth,image,char *,i,int,int *,o
ENCODING,ISencoding,image,char *,i,int,int *,o
COLORMAP_LENGTH,IScolormap_length,image,char *,i,int,int *,o
COLORMAP_ENTRY_SIZE,IScolormap_entry_size,image,char *,i,int,int *,o
COLORMAP,IScolormap,image,char *,i,colormap,colormap_t *,o
PIXELMATRIX,ISpixelmatrix,image,char *,i,char,unsigned char *,o
WINDOW,ISwindow,image,char *,i,int,int,i,int,int,i,int,int,i,int,int,i,image,char *,o
ADD_DESCRIPTION,ISadd_description,image,char *,o,char,char *,i
REPLACE_DESCRIPTION,ISreplace_description,image,char *,o,char,char *,i
DESCRIPTION_LENGTH,ISdescription,image,char *,i,int,int *,o
DESCRIPTION,ISdescription,image,char *,i,char,char *,o
SHOWS,ISshows,image,char *,i,char,char *,i,int,int *,o
CONSTRUCT_IMAGE,ISconstruct_image,int,int,i,int,int,i,int,int,i,int,int,i,int,int,i,int,int,i,
colormap,colormap_t *,i,char,unsigned char *,i,image,char ,o
```

APPENDIX E

CODE FOR PROCESSING INSERT STATEMENT

```
/* ****************************************************************************** */

#include <stdio.h>
#include <malloc.h>
#include "defines.h"
#include "errors.h"
#include "structures.h"

/* ****************************************************************************** */

process_insert(input,output,inbuff,num_entries)

/*
  Called by process_next_sql each time an EXEC SQL INSERT statement is
  identified.
*/

 FILE *input,
     *output;
 char *inbuff;
 int num_entries;

 {
  int i,
     indent = 0,
     inbuff_ndx = 0,
     collecting_values = FALSE,
     end_statement = FALSE;
  char token[MAX_TOKEN + 1],
      outbuff[MAX_LINE+1];
  struct insert_node_struct *insert_node;


  /* create the structure to hold the parsed components of the insert
    statement */
  insert_node=(struct insert_node_struct *) malloc
            (sizeof(struct insert_node_struct));

  /* skip any white space */
  while ((inbuff[indent] == SPACE) ||
       (inbuff[indent] == TAB))
   if (inbuff[indent] == SPACE)
    indent++;
   else
```

```
    indent = indent + 4;

  get_next_token(inbuff,inbuff_ndx,token,&inbuff_ndx);
  to_upper_case(token);

  /* get the table name from the input statement */
  if(strcmp(token,"EXEC") == 0)    /* no label */
    for (i=0; i<4; i++)
      get_next_token(inbuff,inbuff_ndx,token,&inbuff_ndx);
  else    /* lable precedes EXEC SQL */
    for (i=0; i<5; i++)
      get_next_token(inbuff,inbuff_ndx,token,&inbuff_ndx);

  /* token = tablename */

  /* initialize the insert structure */
  insert_node->tablename = (char *) malloc(strlen(token)+1);
  strcpy(insert_node->tablename,token);
  insert_node->col_list = NULL;
  insert_node->expr_list = NULL;

  /* parse the remainder of the insert statement */
  while (!end_statement)
    {
      switch(inbuff[inbuff_ndx]){
        case LEFT_PAREN:
          /* determine the type of list (column or expression)
             encountered and initiate appropriate list processing */
          inbuff_ndx++;
          if (collecting_values)
            {
             process_expr_list(input,insert_node,inbuff,&inbuff_ndx);
             collecting_values = FALSE;
            }
          else
            process_column_list (input,insert_node,inbuff,&inbuff_ndx);
          break;
        case SEMI_COLON:
          /* end of insert statement encountered */
          end_statement = TRUE;
          break;
        case NEW_LINE:
          /* get new line from input file and continue processing */
          fgets(inbuff,MAX_LINE,input);
          inbuff_ndx = 0;
          break;
        case COMMA:
        case TAB:
        case SPACE:
          /* skip white space */
          inbuff_ndx++;
```

```
        break;
      default:
        /* encountered beginning of next token */
        get_next_token(inbuff,inbuff_ndx,token,&inbuff_ndx);
        to_upper_case(token);
        switch(get_token_type(token)){
          case VALUES_CLAUSE:
            /* found beginning of VALUES clause */
            collecting_values = TRUE;
            break;
          case SUBQUERY:
            /* found subquery, not handled by this version of
               the preprocessor */
            break;
          default:
            /* unknown part found in insert statement */
            return INSERT_ERR;
        } /* switch on token type */
      } /* switch on inbuff[inbuff_ndx] */
    } /* while */

  /* bracket preprocessor output so that C compiler will accept any
     declarations that might be generated in the middle of the function */
  clear_buffer(outbuff);
  outbuff[0] = LEFT_BRACKET;
  outbuff[1] = NEW_LINE;
  fputs(outbuff,output);

  /* convert any ISQL expressions to ESQL */
  if (insert_node->expr_list != NULL)
    convert_exprs(output,insert_node,num_entries);

  output_table_name(insert_node,indent,output);

  if (insert_node->col_list != NULL)
    output_columns(insert_node,indent,output);

  if (insert_node->expr_list != NULL)
    output_exprs(insert_node,indent,output);

  /* close bracket of preprocessor output */
  clear_buffer(outbuff);
  outbuff[0] = RIGHT_BRACKET;
  fputs(outbuff,output);

} /* process_insert */
```

```
/* ************************************************************************ */

process_column_list(input,insert_node,inbuff,inbuff_ndx)

/*
   Parses the list of columns from the input file and constructs a list
   of column names, part of an insert node structure. Called by
   process_insert zero or one times, depending on absence or presence
   of column list.
*/

 FILE *input;
 struct insert_node_struct *insert_node;
 char *inbuff;
 int *inbuff_ndx;

 {
  int buff_ndx,
     end_list = FALSE,
     first_column = TRUE;
  char token[MAX_TOKEN + 1];
  struct column *next_column,
             *curr_column;

  buff_ndx = *inbuff_ndx;

  while (!end_list)
   {
    switch(inbuff[buff_ndx]){
     case NEW_LINE:
       /* may encounter new line in middle of column list; have to
          get another line from input file, then continue processing */
       fgets(inbuff,MAX_LINE,input);
       buff_ndx = 0;
       get_next_token(inbuff,buff_ndx,token,&buff_ndx);
       break;
     case RIGHT_PAREN:
       /* signals end of column list */
       end_list = TRUE;
       buff_ndx++;
       break;
     case COMMA:
       /* separates column names, skip */
     case TAB:
     case SPACE:
       /* white space, skip */
       buff_ndx++;
       break;
     default:
      /* beginning of next token */
      get_next_token(inbuff,buff_ndx,token,&buff_ndx);
```

```
      next_column = (struct column *) malloc (sizeof(struct column));
      next_column->column_name = malloc (strlen(token) + 1);
      strcpy(next_column->column_name,token);
      next_column->next = NULL;
      if (first_column)  /* create a new list */
        {
         first_column = FALSE;
         insert_node->col_list = next_column;
        }
      else  /* add to existing list */
        curr_column->next = next_column;
      curr_column = next_column;
     }  /* switch on inbuff[inbuff_ndx]  */
  }
 *inbuff_ndx = buff_ndx;
}
```

```
/* ************************************************************************* */

process_expr_list(input,insert_node,inbuff,inbuff_ndx)

/*
   Parses the list of expressions that follow VALUES in the input file and
   constructs a list of expressions, part of an insert node structure.
   Called by process_insert zero or one times, depending on absence or
   presence of VALUES clause.
*/

 FILE *input;
 struct insert_node_struct *insert_node;
 char *inbuff;
 int *inbuff_ndx;

 {
  int buff_ndx,
      end_list = FALSE,
      first_expr = TRUE;
  char token[MAX_TOKEN + 1];
  struct expr *next_expr,
             *curr_expr;

  buff_ndx = *inbuff_ndx;

  while (!end_list)
   {
    switch(inbuff[buff_ndx]){
     case NEW_LINE:
        /* new line encountered in middle of expression list; get
           another line from input file and continue processing */
        fgets(inbuff,MAX_LINE,input);
        buff_ndx = 0;
        get_next_expr(input,inbuff,buff_ndx,token,&buff_ndx);
        break;
     case RIGHT_PAREN:
        /* end of expression list */
        end_list = TRUE;
        buff_ndx++;
        break;
     case COMMA:
        /* separate expressions, skip */
     case TAB:
     case SPACE:
        /* white space, skip */
        buff_ndx++;
        break;
     default:
       /* beginning of new expression */
       get_next_expr(input,inbuff,buff_ndx,token,&buff_ndx);
```

```
        next_expr = (struct expr *) malloc (sizeof(struct expr));
        next_expr->expression = malloc (strlen(token) + 1);
        strcpy(next_expr->expression,token);
        next_expr->next = NULL;
        if (first_expr)  /* create new list */
          {
           first_expr = FALSE;
           insert_node->expr_list = next_expr;
          }
        else  /* add to existing list */
          curr_expr->next = next_expr;
        curr_expr = next_expr;
      }  /* switch on inbuff[inbuff_ndx]  */
   }
 *inbuff_ndx = buff_ndx;
}
```

```
/* ************************************************************************** */

convert_exprs(output,insert_node,num_entries)

/*
   Checks to see if any input expression represents a call to an external
   function; if so, initiates execution of translation from ISQL to ESQL
   Called by process_insert for each INSERT statement.
*/

 FILE *output;
 struct insert_node_struct *insert_node;
 int num_entries;

 {
  struct expr *next_expr;
  char token[MAX_TOKEN + 1];
  int i,
     expr_num = 0,
     expr_ndx,
     tab_ndx;
  static int translate_num = 1;

  next_expr = insert_node->expr_list;

  /* perform for each expression in the list */
  while (next_expr != NULL)
   {
     expr_ndx = 0;

     /* position at beginning of first token */
     while (is_delimiter(next_expr->expression[expr_ndx]))
      expr_ndx++;

     get_next_token(next_expr->expression,expr_ndx,token,&expr_ndx);

     /* check to see if token represents call to an external function */
     if (is_image_function(token,num_entries,&tab_ndx))
       {
        /* if it is a call to an external function, make the ISQL to
          ESQL translation */
        translate_function_call(output,insert_node,next_expr->expression,
                     expr_ndx,&expr_num,tab_ndx,translate_num);

        /* make sure that position of next_expr pointer is after any
          inserted expression */
        next_expr = insert_node->expr_list;
        for (i=0; i<expr_num; i++)
         next_expr = next_expr->next;

        translate_num++;
```

```
        }

        expr_num++;

        /* move on to next unprocessed incoming expression */
        next_expr = next_expr->next;
    }
}
```

```
/* ************************************************************************ */

translate_function_call(output,insert_node,expr,expr_ndx,expr_num,tab_ndx,translate_num)

/*
   Translate an expression that represents an ESQL call to an external
   function to a series of equivalent ISQL statements that utilize
   corresponding internal functions.  Called by convert_exprs.
*/

 FILE *output;
 struct insert_node_struct *insert_node;
 char *expr;
 int *expr_num,
    expr_ndx,
    tab_ndx,
    translate_num;

 {
  struct ISfunc_struct ISfunc;
  struct ISparam *next_ISparam;
  struct param *next_in_param,
            *return_type;
  char *ptr,
      trans_str[MAX_TOKEN + 1],
      outbuff[MAX_LINE + 1],
      filename[MAX_TOKEN + 1],
      descr[MAX_TOKEN + 1],
      param_name[MAX_TOKEN + 1];
  int i,
     first_param = TRUE,
     outbuff_ndx;


  /* initialize ISfunc structure with name and null pointer */
  ISfunc.name = (char *) malloc (strlen(func_tab[tab_ndx]->ISfunc_name)+1);
  strcpy(ISfunc.name, func_tab[tab_ndx]->ISfunc_name);
  ISfunc.param_list = NULL;

  next_in_param = func_tab[tab_ndx]->in_param_list;

  /* perform for each input parameter associated with internal function */
  while (next_in_param != NULL)
   {
    while (is_delimiter(expr[expr_ndx]))
     expr_ndx++;

    /* get next parameter from external function expression */
    get_next_token(expr,expr_ndx,param_name,&expr_ndx);

    if (first_param)  /* create new list of ISparams */
```

```
      {
       first_param = FALSE;
       ISfunc.param_list = (struct ISparam *) malloc (sizeof(struct ISparam));
       next_ISparam = ISfunc.param_list;
      }
    else /* add to existing ISparam list */
      {
       next_ISparam->next = (struct ISparam *) malloc (sizeof(struct ISparam));
       next_ISparam = next_ISparam->next;
      }

    /* load data into ISparam structure */
    next_ISparam->name = (char *) malloc (strlen(param_name) + 1);
    strcpy(next_ISparam->name, param_name);
    next_ISparam->next = NULL;

    /* if param is of type image, convert to filename and description
      name pair and change one parameter to 2 parameters */
    if (strcmp(next_in_param->param_type,"image") == 0)
      {
       /* convert to filename/description pair */
       ISget_names(next_ISparam->name,filename,descr);

       /* substitute filename (_f) for incoming image name */
       next_ISparam->name = (char *) malloc (strlen(filename) + 1);
       strcpy(next_ISparam->name, filename);

       /* create new ISparam struct and insert description (_d) param */
       next_ISparam->next = (struct ISparam *) malloc (sizeof(struct ISparam));
       next_ISparam = next_ISparam->next;
       next_ISparam->name = (char *) malloc (strlen(filename) + 1);
       strcpy(next_ISparam->name,descr);
       next_ISparam->next = NULL;
      } /* if */

    next_in_param = next_in_param->next;
  } /* while */

/* output include statement for definitions */
fputs("#include \"defines.h\"\n\n",output);

/*  output opening statement of declaration section */
fputs("EXEC SQL BEGIN DECLARE SECTION;\n", output);

clear_buffer(outbuff);
/* indent for declaration */
for (outbuff_ndx=0; outbuff_ndx < ADD_INDENT; outbuff_ndx++)
  outbuff[outbuff_ndx] = SPACE;

/* set up to add return type to list of ISparams */
return_type = func_tab[tab_ndx]->return_type;
```

```
next_ISparam->next = (struct ISparam *) malloc (sizeof(struct ISparam));
next_ISparam = next_ISparam->next;

/* if the return type is image, add two ISparams to list and modify
   corresponding item in column list */
if (strcmp(return_type->param_type,"image") == 0)
  {
    /* translate corresponding column name to _f/_d pair */
    translate_column_name(insert_node, *expr_num);

    /* all variables created to hold filename are a concatenation of
       "ISfn" and the current translate_num to ensure a unique
       variable name */
    clear_token(param_name);
    strcat(param_name,"ISfn");
    sprintf(trans_str,"%4d",translate_num);
    fill_space_with_zero(trans_str);
    strcat(param_name,trans_str);
    next_ISparam->name = (char *) malloc (strlen (param_name) + 1);
    strcpy (next_ISparam->name,param_name);
    /* replace the expression with the corresponding return variable name */
    translate_expr (insert_node,*expr_num,param_name);
    /* output filename variable declaration */
    put_next_token(return_type->param_decl,outbuff,outbuff_ndx,&outbuff_ndx);
    put_next_token(next_ISparam->name,outbuff,outbuff_ndx,&outbuff_ndx);
    put_next_token("[FILE_NAME_LENGTH + 1]",outbuff,outbuff_ndx,&outbuff_ndx);
    outbuff[outbuff_ndx++] = SEMI_COLON;
    outbuff[outbuff_ndx] = NEW_LINE;
    fputs (outbuff,output);

    /* all variables created to hold description are a concatenation of
       "ISdescr" and the current translate_num to ensure a unique
       variable name */
    clear_buffer(outbuff);
    /* indent for declaration */
    for (outbuff_ndx=0; outbuff_ndx < ADD_INDENT; outbuff_ndx++)
      outbuff[outbuff_ndx] = SPACE;

    next_ISparam->next = (struct ISparam *) malloc (sizeof(struct ISparam));
    next_ISparam = next_ISparam->next;

    clear_token(param_name);
    strcat(param_name,"ISdescr");
    sprintf(trans_str,"%4d",translate_num);
    fill_space_with_zero(trans_str);
    strcat(param_name,trans_str);
    next_ISparam->name = (char *) malloc (strlen(param_name) + 1);
    strcpy (next_ISparam->name,param_name);
    /* insert descr param after filename param in expression list */
    insert_expr(insert_node,expr_num,param_name);
    /* output description variable declaration */
```

```
    put_next_token(return_type->param_decl,outbuff,outbuff_ndx,&outbuff_ndx);
    put_next_token(next_ISparam->name,outbuff,outbuff_ndx,&outbuff_ndx);
    put_next_token("[MAX_DESCR + 1]",outbuff,outbuff_ndx,&outbuff_ndx);
    outbuff[outbuff_ndx++] = SEMI_COLON;
    outbuff[outbuff_ndx] = NEW_LINE;
    fputs (outbuff,output);
   }
  else /* return type not of type image */
   {
    /* all variables created to hold other than filename or description
       are a concatenation of "ISvar" and the current translate_num to
       ensure a unique variable name */
    clear_token(param_name);
    strcat(param_name,"ISvar");
    sprintf(trans_str,"%4d",translate_num);
    fill_space_with_zero(trans_str);
    strcat(param_name,trans_str);
    next_ISparam->name = (char *) malloc (strlen (param_name) + 1);
    strcpy (next_ISparam->name,param_name);
    /* replace the expression with the corresponding variable name */
    translate_expr (insert_node,*expr_num,param_name);
    /* output variable name declaration */
    put_next_token(return_type->param_decl,outbuff,outbuff_ndx,&outbuff_ndx);
    put_next_token(next_ISparam->name,outbuff,outbuff_ndx,&outbuff_ndx);
    outbuff[outbuff_ndx++] = SEMI_COLON;
    outbuff[outbuff_ndx] = NEW_LINE;
    fputs (outbuff,output);
   }

  /* output closing statement of declaration section */
  fputs("EXEC SQL END DECLARE SECTION;\n\n",output);

  clear_buffer(outbuff);
  fputs(outbuff,output);

  /*  output internal function call that replaces internal function call */
  output_ISfunc_call(output,&ISfunc);
  fputs(outbuff,output);
}
```

```
/* ************************************************************************** */

translate_column_name(insert_node,expr_num)

/*
   Replaces column name occurring at position number expr_num in column_list
   of insert_node with its _f/_d pair.
*/

 struct insert_node_struct *insert_node;
 int expr_num;

 {
  int i;
  struct column *curr_col,
              *new_col;
  char filename[MAX_TOKEN + 1],
      descr[MAX_TOKEN + 1];
  curr_col = insert_node->col_list;


  for (i=0; i<expr_num; i++)
    curr_col = curr_col->next;

  ISget_names(curr_col->column_name,filename,descr);

  curr_col->column_name = (char *) malloc (strlen(filename) + 1);
  strcpy(curr_col->column_name,filename);

  new_col = (struct column *) malloc (sizeof(struct column));
  new_col->column_name = (char *) malloc (strlen(descr) + 1);
  strcpy (new_col->column_name, descr);

  new_col->next = curr_col->next;
  curr_col->next = new_col;
 }
```

```
/* ************************************************************************ */

translate_expr(insert_node,expr_num,param)

/*
   Replaces expression occurring at position number expr_num in expr_list
   of insert_node with param. Called by translate_function_call.
*/

 struct insert_node_struct *insert_node;

 int expr_num;
 char *param;

 {
  int i;

  struct expr *curr_expr;

  curr_expr = insert_node->expr_list;

  for (i=0; i<expr_num; i++)
    curr_expr = curr_expr->next;

  curr_expr->expression = (char *) malloc (strlen(param) + 1);
  strcpy(curr_expr->expression,param);
 }
```

```
/* ************************************************************************ */

insert_expr(insert_node,expr_num,param)

/*
   Inserts param into expr_list of insert_node following expression
   occurring at position expr_num.  Called by translate_function_call.
*/

 struct insert_node_struct *insert_node;
 int *expr_num;
 char *param;

 {
  int i;
  struct expr *curr_expr,
            *new_expr;

  curr_expr = insert_node->expr_list;

  for (i=0; i<*expr_num; i++)
    curr_expr = curr_expr->next;

  new_expr = (struct expr *) malloc (sizeof(struct expr));
  new_expr->expression = (char *) malloc (strlen(param) + 1);
  strcpy(new_expr->expression,param);

  new_expr->next= curr_expr->next;
  curr_expr->next = new_expr;

  (*expr_num)++;
 }
```

```
/* ************************************************************************ */

output_table_name(insert_node,indent,output)

/*
  Sends the first nonvariable part of the insert statement and the table
  name to the output file.
*/

 struct insert_node_struct *insert_node;
 int indent;
 FILE *output;

 {
  int outbuff_ndx;
  char outbuff[MAX_LINE + 1];

  clear_buffer(outbuff);
  for (outbuff_ndx=0; outbuff_ndx<indent; outbuff_ndx++)
    outbuff[outbuff_ndx] = SPACE;
  put_next_token("EXEC SQL INSERT INTO ",outbuff,outbuff_ndx,&outbuff_ndx);
  put_next_token(insert_node->tablename,outbuff,outbuff_ndx,&outbuff_ndx);
  outbuff[outbuff_ndx] = NEW_LINE;
  fputs(outbuff,output);
 }
```

```
/* ************************************************************************ */

output_columns(insert_node,indent,output)

/*
   Sends the formatted column list to the output file.
*/

 struct insert_node_struct *insert_node;
 int indent;
 FILE *output;

 {
  int outbuff_ndx = 0;
  char outbuff[MAX_LINE + 1];
  struct column *next_column;

  clear_buffer(outbuff);
  for (outbuff_ndx=0; outbuff_ndx<indent+ADD_INDENT; outbuff_ndx++)
    outbuff[outbuff_ndx] = SPACE;
  outbuff[outbuff_ndx++] = LEFT_PAREN;
  next_column = insert_node->col_list;
  while (next_column != NULL)
    {
     put_next_token(next_column->column_name,outbuff,outbuff_ndx,&outbuff_ndx);
     put_next_token(", ",outbuff,outbuff_ndx,&outbuff_ndx);
     next_column = next_column->next;
    }
  outbuff_ndx = outbuff_ndx - 2;
  outbuff[outbuff_ndx++] = RIGHT_PAREN;
  outbuff[outbuff_ndx++] = NEW_LINE;
  fputs (outbuff,output);
 }
```

```
/* ************************************************************************** */

output_exprs(insert_node,indent,output)

/*
   Sends the formatted expression list to the output file.
*/

 struct insert_node_struct *insert_node;
 int indent;
 FILE *output;

 {
  int outbuff_ndx = 0;
  char outbuff[MAX_LINE + 1];
  struct expr *next_expr;

  clear_buffer(outbuff);
  for (outbuff_ndx=0; outbuff_ndx<indent+ADD_INDENT; outbuff_ndx++)
    outbuff[outbuff_ndx] = SPACE;
  put_next_token("VALUES ",outbuff,outbuff_ndx,&outbuff_ndx);
  outbuff[outbuff_ndx++] = LEFT_PAREN;
  next_expr = insert_node->expr_list;
  while (next_expr != NULL)
    {
     outbuff[outbuff_ndx++] = COLON;
     put_next_token(next_expr->expression,outbuff,outbuff_ndx,&outbuff_ndx);
     put_next_token(", ",outbuff,outbuff_ndx,&outbuff_ndx);
     next_expr = next_expr->next;
    }
  outbuff_ndx = outbuff_ndx - 2;
  outbuff[outbuff_ndx++] = RIGHT_PAREN;
  outbuff[outbuff_ndx++] = SEMI_COLON;
  outbuff[outbuff_ndx++] = NEW_LINE;
  fputs (outbuff,output);
 }
```

APPENDIX F

SAMPLE INPUT FILE

```
#include <stdio.h>
#include <sys/types.h>
#include <pixrect/pixrect_hs.h>
#include "defines.h"

main()
{
 struct pixrect *pr;
 colormap_t cm;
 FILE *input;
 char *input_filename = "testimage.1";

 EXEC SQL INCLUDE SQLCA;

 EXEC SQL BEGIN DECLARE SECTION;
  int id;
 EXEC SQL END DECLARE SECTION;

 if ((input = fopen (input_filename,"r")) == NULL)
  {
   printf ("Cannot open file %s\n", input_filename);
   exit (1);
  }

 if ((pr = pr_load (input, &cm)) == NULL)
  {
   printf ("cannot load pixrect from file %s\n", input_filename);
   fclose (input);
   exit (1);
  }

 fclose (input);

 EXEC SQL CONNECT "imagedb";

 printf ("Please enter the id number for the new photo: ");
 scanf ("%d",&id);
 while (getchar() != NEW_LINE)
  ;

 EXEC SQL INSERT INTO image
  (i_id, i_image)
  VALUES (:id, IMAGE_FROM_PIXRECT(pr, &cm));
```

```
 EXEC SQL DISCONNECT;
}
```

APPENDIX G

SAMPLE OUTPUT FILE

```
#include <stdio.h>
#include <sys/types.h>
#include <pixrect/pixrect_hs.h>
#include "defines.h"

main()
{
  struct pixrect *pr;
  colormap_t cm;
  FILE *input;
  char *input_filename = "testimage.1";

  EXEC SQL INCLUDE SQLCA;

  EXEC SQL BEGIN DECLARE SECTION;
    int id;
  EXEC SQL END DECLARE SECTION;

  if ((input = fopen (input_filename,"r")) == NULL)
    {
      printf ("Cannot open file %s\n", input_filename);
      exit (1);
    }

  if ((pr = pr_load (input, &cm)) == NULL)
    {
      printf ("cannot load pixrect from file %s\n", input_filename);
      fclose (input);
      exit (1);
    }

  fclose (input);

  EXEC SQL CONNECT "imagedb";

  printf ("Please enter the id number for the new photo: ");
  scanf ("%d",&id);
  while (getchar() != NEW_LINE)
    ;

{
#include "defines.h"

EXEC SQL BEGIN DECLARE SECTION;
```

```
   char ISfn0001[FILE_NAME_LENGTH + 1];
   char ISdescr0001[MAX_DESCR + 1];
EXEC SQL END DECLARE SECTION;

ISimage_from_pixrect(pr,&cm,ISfn0001,ISdescr0001);

 EXEC SQL INSERT INTO image
    (i_id, i_image_f, i_image_d)
    VALUES (:id, :ISfn0001, :ISdescr0001);
}
 EXEC SQL DISCONNECT;
}
```

APPENDIX H

UTILITY SOFTWARE

```
/* ************************************************************************** */

#include <ctype.h>
#include <stdio.h>
#include "defines.h"
#include "errors.h"
#include "structures.h"

/* ************************************************************************** */

clear_buffer(buffer)

/*
  Fills the buffer with TERMINAL's (\0).
*/

  char *buffer;

  {
   int i;

   for (i=0; i<MAX_LINE+1; i++)
     buffer[i] = TERMINAL;
  }
```

```
/* ************************************************************************** */

clear_token(token)

/*
  Fills the token with TERMINAL's (\0).
*/

  char *token;

  {
   int i;

   for (i=0; i<MAX_TOKEN+1; i++)
     token[i] = TERMINAL;
  }
```

```
/* ************************************************************************* */

fill_space_with_zero(str)

/*
  Fills blanks in the str with zeros.
*/

  char *str;

  {
   char *ptr;

   for (ptr=str; *ptr; ptr++)
     if (*ptr == SPACE)
       *ptr = '0';
  }
```

```
/* ************************************************************************** */

get_next_token(inbuff,buff_ndx,token,new_ndx)

/*
  Takes the line sent to it and removes the token starting at buff_ndx.
  Returns inbuff unchanged, but new_ndx contains the new position within
  the line for the next operation.
*/

 char inbuff[MAX_LINE + 1],
     token[MAX_TOKEN + 1];
 int buff_ndx,
    *new_ndx;

 {
  int token_ndx = 0;

  /* Skips intervening white space. */
  while ((inbuff[buff_ndx] == SPACE) ||
        (inbuff[buff_ndx] == TAB))
    buff_ndx++;

  /* Reads characters into the token until a blank or a new line
     character is encountered. */
  while (!is_delimiter(token[token_ndx] = inbuff[buff_ndx]))
    {
     buff_ndx++;
     token_ndx++;
    }
  /* Place the terminating null character at the end of the token. */
  token[token_ndx] = TERMINAL;
  *new_ndx = buff_ndx;
 }
```

```
/* ************************************************************************** */

get_next_expr(input,inbuff,buff_ndx,token,new_ndx)

/*
  Takes the line sent to it and removes the token starting at buff_ndx.
  Returns inbuff unchanged, but new_ndx contains the new position within
  the line for the next operation.
*/

 FILE *input;
 char inbuff[MAX_LINE + 1],
     token[MAX_TOKEN + 1];
 int buff_ndx,
    *new_ndx;

 {
  int end_expr = FALSE,
      num_left_parens = 0,
      token_ndx = 0;

  /* Skips intervening white space. */
  while ((inbuff[buff_ndx] == SPACE) ||
        (inbuff[buff_ndx] == TAB))
    buff_ndx++;

  /* Reads characters into the token until a delimiter is encountered. */
  while (!end_expr)
    {

    switch(inbuff[buff_ndx]){
      case RIGHT_PAREN:
         if (!num_left_parens)
           end_expr = TRUE;
         else
           num_left_parens--;
         break;
      case LEFT_PAREN:
         num_left_parens++;
         break;
      case NEW_LINE:
      case TERMINAL:
         fgets (inbuff,MAX_LINE,input);
         buff_ndx=0;
         break;
      case COMMA:
         if (!num_left_parens)
           end_expr = TRUE;
         break;
      case COLON:
      case TAB:
```

```
    case SPACE:
      buff_ndx++;
      break;
    default:
      ;

  }
    if (!end_expr)
      {
      token[token_ndx] = inbuff[buff_ndx];
      buff_ndx++;
      token_ndx++;
      }

  }
 /* Place the terminating null character at the end of the token. */
 token[token_ndx] = TERMINAL;
 *new_ndx = buff_ndx;
}
```

```
/* ************************************************************************** */

get_operation_type(operation)

/*
   Evalutes operation token and returns numeric equivalent.
*/

 char *operation;

 {
  if (strcmp(operation,"ABORT") == 0)
   return ABORT_OP;
  else if (strcmp(operation,"BEGIN") == 0)
   return BEGIN_OP;
  else if (strcmp(operation,"CLOSE") == 0)
   return CLOSE_OP;
  else if (strcmp(operation,"COPY") == 0)
   return COPY_OP;
  else if (strcmp(operation,"CONNECT") == 0)
   return CONNECT_OP;
  else if (strcmp(operation,"CREATE") == 0)
   return CREATE_OP;
  else if (strcmp(operation,"DECLARE") == 0)
   return DECLARE_OP;
  else if (strcmp(operation,"DELETE") == 0)
   return DELETE_OP;
  else if (strcmp(operation,"DISCONNECT") == 0)
   return DISCONNECT_OP;
  else if (strcmp(operation,"DROP") == 0)
   return DROP_OP;
  else if (strcmp(operation,"END") == 0)
   return END_OP;
  else if (strcmp(operation,"FETCH") == 0)
   return FETCH_OP;
  else if (strcmp(operation,"HELD") == 0)
   return HELD_OP;
  else if (strcmp(operation,"INCLUDE") == 0)
   return INCLUDE_OP;
  else if (strcmp(operation,"INSERT") == 0)
   return INSERT_OP;
  else if (strcmp(operation,"MODIFY") == 0)
   return MODIFY_OP;
  else if (strcmp(operation,"OPEN") == 0)
   return OPEN_OP;
  else if (strcmp(operation,"PRINT") == 0)
   return PRINT_OP;
  else if (strcmp(operation,"RELOCATE") == 0)
   return RELOCATE_OP;
  else if (strcmp(operation,"SAVE") == 0)
   return SAVE_OP;
```

```
  else if (strcmp(operation,"SAVEPOINT") == 0)
    return SAVEPOINT_OP;
  else if (strcmp(operation,"SELECT") == 0)
    return SELECT_OP;
  else if (strcmp(operation,"SET") == 0)
    return SET_OP;
  else if (strcmp(operation,"UPDATE") == 0)
    return UPDATE_OP;
  else if (strcmp(operation,"WHENEVER") == 0)
    return WHENEVER_OP;
  else
    return NULL;
}
```

```
/* ************************************************************************* */

get_token_type(token)

/*
  Evalutes parsed token and returns numeric equivalent.
*/

 char token[MAX_TOKEN +1];

 {
  int i;

  if (strcmp(token,"VALUES") == 0)
    return VALUES_CLAUSE;
  else if (strcmp(token,"SELECT") == 0)
    return SUBQUERY;
  else
    return INSERT_ERR;
 }
```

```
/* ******************************************************************** */

ISget_names(image_attribute,filename_attribute,descr_attribute)

/*
   Convert IMAGE name to filename (_f) and description (_d) pair.
*/

  char *image_attribute,  /* input */
       *filename_attribute, /* output */
       *descr_attribute;  /* output */

{
  int i;

  for (i=0; i<MAX_TOKEN; i++)
    filename_attribute[i] = descr_attribute[i] = TERMINAL;

  for (i=0; ((image_attribute[i] != TERMINAL) && (i<MAX_SQL_NAME)); i++)
    filename_attribute[i] = descr_attribute[i] = image_attribute[i];
  filename_attribute[i] = descr_attribute[i] = UNDERSCORE;
  filename_attribute[i+1] = 'f';
  descr_attribute[i+1] = 'd';
}
```

```
/* ************************************************************************** */

is_delimiter(ch)

/*
   Determines if the character submitted is a delimiter or not.
   Returns TRUE or FALSE.
*/

 char ch;

 {
  switch(ch)
    {
     case SPACE       :
     case TAB         :
     case NEW_LINE    :
     case LEFT_PAREN  :
     case RIGHT_PAREN :
     case COMMA       :
     case SEMI_COLON  :
     case COLON       :
        return TRUE;
        break;
     default   :
        return FALSE;
    }
 }
```

```
/* ******************************************************************************** */

is_image_function(token,num_entries,tab_ndx)

/*
   Searches function table to determine if token represents a call
   to an external IMAGE function.  Returns TRUE or FALSE.
*/

 char *token;
 int num_entries,
    *tab_ndx;

 {
  int found = FALSE;

  *tab_ndx = 0;
  while ((!found) && (*tab_ndx < num_entries))
   if (strcmp(token,func_tab[*tab_ndx]->func_name) == 0)
     found = TRUE;
   else
     (*tab_ndx)++;
  return found;
 }
```

```
/* ************************************************************************** */

output_ISfunc_call(output,ISfunc)

/*
   Outputs formatted internal function call.
*/

 FILE *output;
 struct ISfunc_struct *ISfunc;

{
 char outbuff[MAX_LINE + 1];
 int outbuff_ndx = 0;
 struct ISparam *next_param;

 clear_buffer(outbuff);

 put_next_token(ISfunc->name,outbuff,outbuff_ndx,&outbuff_ndx);
 outbuff[outbuff_ndx++] = LEFT_PAREN;

 next_param = ISfunc->param_list;
 while (next_param != NULL)
   {
    put_next_token (next_param->name,outbuff,outbuff_ndx,&outbuff_ndx);
    outbuff[outbuff_ndx++] = COMMA;
    next_param = next_param->next;
   }

 outbuff[outbuff_ndx-1] = RIGHT_PAREN;
 outbuff[outbuff_ndx++] = SEMI_COLON;
 outbuff[outbuff_ndx++] = NEW_LINE;
 outbuff[outbuff_ndx] = NEW_LINE;
 fputs (outbuff,output);
}
```

```
/* ************************************************************************ */

put_next_token(token,outbuff,buff_ndx,new_ndx)

/*
   Writes a token to the output buffer starting at buff_ndx, and returns the
   new position within the output buffer in new_ndx.
*/

 char *token,
      *outbuff;
 int buff_ndx,
     *new_ndx;

 {
  int token_ndx;

  token_ndx = 0;

  /* Writes each character in the token to the output buffer until the
     terminating null is encountered. */
  while (token[token_ndx] != TERMINAL)
   {
    outbuff[buff_ndx] = token[token_ndx];
    buff_ndx++;
    token_ndx++;
   }
  *new_ndx = buff_ndx;
 }
```

```
/* ************************************************************************** */

to_upper_case(token)

/*
  Converts any lowercase characters in token to uppercase.
*/

 char token[MAX_TOKEN + 1];

 {
  int token_ndx;

  token_ndx = 0;
  while (token[token_ndx] != TERMINAL)
    {
     if (islower(token[token_ndx]))
       token[token_ndx] = toupper(token[token_ndx]);
       token_ndx++;
    }
 }
```

APPENDIX I

ERROR CODES

```
#define FOPEN_ERR 300
#define PR_DUMP_ERR 301
#define PR_LOAD_ERR 302
#define PR_LOAD_HEADER_ERR 303
#define PR_LOAD_COLORMAP_ERR 304
#define MEM_CREATE_ERR 305
#define PR_REGION_ERR 306
#define PR_DUMP_HEADER_ERR 307
#define PR_DUMP_IMAGE_ERR 308
#define FWRITE_ERR 309
#define FREAD_ERR 310
#define MALLOC_ERR 311

#define NO_COLORMAP_ERR 200
#define NO_WIDTH_ERR 201
#define NO_HEIGHT_ERR 202
#define NO_DEPTH_ERR 203
#define INVALID_ENCODING_ERR 204
#define INVALID_RASTER_TYPE_ERR 205
#define INVALID_MAP_TYPE_ERR 206
#define IMAGE_FROM_PIXRECT_ERR 207
#define FUNC_TAB_ERR 208
#define PARAM_ERR 209
#define INSERT_ERR 210
#define DESCR_TOO_LONG_ERR 211
#define NO_COLORMAP_ENTRY_SIZE_ERR 212
#define MAPLENGTH_IN_BYTE_WARNING 213
#define INVALID_WINDOW_PARAMS 214
#define NO_WINDOW_OVERLAP 215
#define WINDOW_WRONG_DEPTH_ERR 216

#define ERROR_FREE 0
```

LIST OF REFERENCES

Enbody, Diane M., Analysis of Existing Advanced Data Models and Their Applicability as a Model for a Multimedia Database Management System, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.

Meyer-Wegener, K., V. Y. Lum, and C. T. Wu, Image Database Management in a Multimedia System, Naval Postgraduate School report no. NPS52-88-024, August 1988.

Ong, J., D. Fogg, and M. Stonebraker, "Implementation of DATA Abstraction in the Relational Database System INGRES," ACM SIGMOD Record, v. 14, pp. 1-14, March 1984.

Sawyer, G., Managing Sound In a Relational Multimedia Database System, Master's Thesis, Naval Postgradaute School, Monterey, California, December 1988.

Tang, G. Y., "A Logical Data Organization for the Integrated Database of Pictures and Alphanumeric Data," Proc. IEEE Workshop on Picture Data Description and Management August 27-28, 1980, pp. 158-164, August 1980.

BIBLIOGRAPHY

Lum, V. Y., C. T. Wu, and D. K. Hsiao, Integrating Advanced Techniques into Multimedia DBMS, Naval Postgraduate School report no. NPS52-87-050, November 1987.

Meyer-Wegener, K., V. Y. Lum, and C. T. Wu, Managing Multimedia Data - An Exploration, Naval Postgraduate School report no. NPS52-88-010, March 1988.

Meyer-Wegener, K., Extending an SQL Interface with the Data Type IMAGE, Naval Postgraduate School design document, August 1988.

Relational Technology Inc., INGRES/Embedded SQL Companion Guide for C, Release 5.0, UNIX, August 1986.

Relational Technology Inc., INGRES/Embedded SQL User's Guide and Reference Manual, Release 5.0, UNIX, August 1986.

Relational Technology Inc., INGRES/SQL Reference Manual, Release 5.0, UNIX, August 1986.

Sun Microsystems, Inc., Pixrect Reference Manual, Part No: 800-1254-03 Revision A, 17 February 1986.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 0142 — 2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Associate Professor C. Thomas Wu, Code 52Wq — 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Klaus Meyer-Wegener — 1
   Universitaet Kaiserslautern
   Fachbereich Informatik
   Postfach 30 49
   6750 Kaiserslautern
   West Germany

5. Professor Vincent Y. Lum, Code 52Lu — 10
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Mr. Albert Wong, Code 52 — 1
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5000

7. LCDR Gregory R. Sawyer, USN — 1
   Attack Squadron Forty-Two
   U.S. Naval Air Station Oceana
   Virginia Beach, Virginia 23460

8. LT Diane M. Enbody — 1
   4693 Blue Pine Circle
   Lake Worth, Florida 33463

9. LT Cathy A. Thomas — 2
   13968 Stoney Gate Pl.
   San Diego, California 92128

Thesis
T385
c.1

Thomas
A program interface prototype for a multi-media database incorporating images.